

AD-A243 189



2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
DEC 05 1991
S D D



THESIS

THE DEVELOPMENT OF USER INTERFACE TOOLS
FOR THE
COMPUTER AIDED PROTOTYPING SYSTEM

by

Mary Ann Cummings

December, 1990

Thesis Advisors:

Luqi
Patrick D. Barnes

Approved for public release; distribution is unlimited.

91-16594

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) 52	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER CCR-8710737	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation/NSWC	8b. OFFICE SYMBOL (if applicable)	10. SOURCE OF FUNDING NUMBERS	
8c. ADDRESS (City, State, and ZIP Code) Washington D.C. 20550/ Dahlgren, VA 22448		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) The Development of User Interface Tools for the Computer Aided Prototyping System			
12. PERSONAL AUTHOR(S) Cummings, Mary Ann			
13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day) December 1990	15. PAGE COUNT 344
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	Prototyping Language, User Interface, Rapid Prototyping, Graphic Editors, Computer Aided Design, Ada, User Interface Toolkit, Real-Time System	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
<p>The Computer Aided Prototyping System (CAPS) was created to rapidly prototype real-time systems in order to determine if the system requirements can be met early in the development cycle. CAPS consists of several software tools that automatically generate an executable Ada model of the proposed system from a given specification. This thesis describes the development of a user interface for CAPS. The user interface supports the design, modification and execution of the software prototype throughout the entire prototyping life cycle. It makes use of X Windows and advanced windowing techniques and allows the user to run the tools concurrently. The user interface incorporates a separate tool interface which controls the interaction between the CAPS tools and the user interface. The graphic editor uses advanced graphics capabilities to give the user more flexibility in editing a graphical representation of the prototype. It automatically produces a formal representation of the prototype to be used by the other tools in CAPS.</p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Luqi		22b. TELEPHONE (Include Area Code) (408) 646-2735	22c. OFFICE SYMBOL 52Lq

Approved for public release; distribution is unlimited.

**THE DEVELOPMENT OF USER INTERFACE TOOLS FOR
THE COMPUTER AIDED PROTOTYPING SYSTEM**

by

Mary Ann Cummings
Civilian, Naval Surface Warfare Center
B.S., James Madsion University, 1984

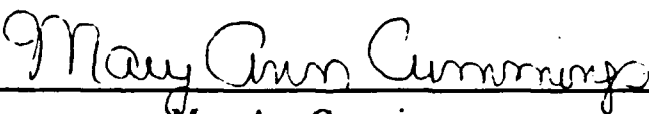
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the


NAVAL POSTGRADUATE SCHOOL
December 1990

Author:




Mary Ann Cummings

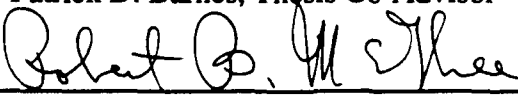
Approved By:



Luqi, Thesis Advisor



Patrick D. Barnes, Thesis Co-Advisor



Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

The Computer Aided Prototyping System (CAPS) was created to rapidly prototype real-time systems in order to determine if the system requirements can be met early in the development cycle. CAPS consists of several software tools that automatically generate an executable Ada model of the proposed system from a given specification. This thesis describes the development of a user interface for CAPS.

The user interface supports the design, modification and execution of the software prototype throughout the entire prototyping life cycle. It makes use of X Windows and advanced windowing techniques and allows the user to run the tools concurrently. The user interface also incorporates a separate tool interface which controls the interaction between the CAPS tools and the user interface.

The graphic editor uses advanced graphics capabilities to give the user more flexibility in editing a graphical representation of the prototype. It automatically produces a formal representation of the prototype to be used by the other tools in CAPS.

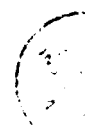
Accession For

J

NHS CRAE
DND LND
JAN 20 1972
JAN 20 1972

F
DND LND

A-1



THESIS DISCLAIMER

Ada is a registered trademark of the United States Government, Ada Joint Program Office.

X Window System is a registered trademark of the Massachusetts Institute of Technology (MIT).

SUN is a registered trademark of Sun Microsystems.

UNIX is a registered trademark of AT&T.

InterViews is a registered trademark of Stanford University.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	PROBLEM STATEMENT	1
B.	SCOPE	2
C.	RESEARCH APPROACH	2
D.	ORGANIZATION	4
II.	BACKGROUND	5
A.	DEVELOPMENT OF LARGE REAL-TIME SYSTEMS	5
B.	THE COMPUTER AIDED PROTOTYPING SYSTEM	7
1.	Prototype Construction Using CAPS	8
a.	Edit	8
b.	Search for Reusable Components	9
c.	Translate	10
d.	Compile and Execute	10
e.	Store Prototype Representations	10
C.	INTERVIEWS: A USER INTERFACE TOOLKIT	11
D.	IDRAW: INTERVIEWS DRAWING EDITOR	13
III.	REQUIREMENTS SPECIFICATION	15
A.	CAPS INTERFACE	15
1.	The Environment Model	17
2.	The Behavioral Model	18
3.	Principles of User Interface Design	21
B.	GRAPHIC EDITOR	21

1.	The Environment Model	21
2.	The Behavioral Model.....	22
IV.	ARCHITECTURAL DESIGN	24
A.	USER INTERFACE	26
1.	Informal Solution	26
2.	Formal Solution.....	26
3.	Implementation Strategy	28
B.	TOOL INTERFACE.....	30
1.	Informal Solution	30
2.	Formal Solution.....	30
3.	Implementation Strategy	32
C.	GRAPHIC EDITOR	33
1.	Analyze Idraw	33
2.	Changes	40
a.	Present Prototype Names To User	40
b.	Remove Unused Commands and Tools.....	41
c.	Add Internal Representation of DFD.....	43
d.	Modify Existing Commands and Tools	46
e.	Add New Tools.....	48
f.	Add Ability to Rebuild DFD Data Structure	51
g.	Add Help Information.....	52
V.	USERS MANUAL	53
A.	INTRODUCTION	53
B.	GETTING STARTED	53
C.	CAPS CONCEPTS	54
1.	General	54

2.	Graphic Editor Concepts	54
D.	USING CAPS	55
E.	USING THE GRAPHIC EDITOR	57
1.	Graphic Editor User Interface	57
2.	Tools.....	57
a.	Select.....	57
b.	Move	57
c.	Modify	58
d.	Specify	58
e.	Streams.....	58
f.	Constraints	59
g.	Decompose.....	59
h.	Comment.....	59
i.	Label	59
j.	MET	60
k.	Latency.....	60
l.	Data Flow.....	60
m.	Operator	60
3.	Commands.....	60
a.	Prototype.....	61
b.	Edit.....	61
c.	Font	61
d.	Pattern	61
e.	Color	61
f.	Align	61
g.	Options.....	62

F.	USING OTHER CAPS FUNCTIONS	62
1.	Search	62
2.	Translate	62
3.	Compile	62
4.	Execute	63
VI.	CONCLUSIONS AND RECOMMENDATIONS	64
A.	SUMMARY	64
B.	RECOMMENDATIONS FOR FURTHER WORK	65
1.	Tool Interface	65
a.	Integrating Interface Functions	65
b.	Integrating the Design Database	65
2.	Graphic Editor	66
a.	Syntax Directed Editor	66
b.	Consistency Checks	67
c.	Modify Tool	67
d.	Abstract Data Types	68
C.	CONCLUSIONS	68
APPENDIX A	CAPS INTERFACE ESSENTIAL MODEL	69
APPENDIX B	PSDL GRAMMAR	79
APPENDIX C	CAPS INTERFACE PROGRAMMERS MANUAL	84
A.	CAPS INTERFACE FILES	84
1.	User Interface	84
2.	Design Database	84
3.	Tool Interface	84
B.	GUIDELINES FOR FUTURE CHANGES	85
1.	User Interface	85

	2.	Design Database.....	85
	3.	Tool Interface.....	86
	C.	CODE.....	87
APPENDIX D		GRAPHIC EDITOR PROGRAMMERS MANUAL.....	110
	A.	FILES.....	110
	B.	GUIDELINES FOR FUTURE CHANGES.....	113
	1.	PSDL Updates.....	113
	2.	Writing Files.....	114
	3.	Adding New Tools.....	114
	4.	Adding New Commands.....	114
	5.	Adding New DFD Components.....	115
	6.	Adding New X Resources.....	115
	6.	Adding Syntax Directed Editors.....	115
	C.	CODE.....	116
		LIST OF REFERENCES.....	327
		BIBLIOGRAPHY.....	329
		INITIAL DISTRIBUTION LIST.....	330

LIST OF FIGURES

Figure 2.1	Classical Project Life Cycle.....	6
Figure 2.2	Prototyping Life Cycle.....	7
Figure 2.3	Interactor Class Hierarchy	12
Figure 2.4	Graphics Class Hierarchy	12
Figure 2.5	Idraw: InterViews Drawing Editor	13
Figure 3.1	The CAPS Environment	16
Figure 3.2	CAPS Interface Context Diagram.....	17
Figure 3.3	CAPS Interface Data Flow Diagram	19
Figure 3.4	User Interface Data Flow Diagram.....	20
Figure 3.5	Graphic Editor Context Diagram.....	22
Figure 3.6	Graphic Editor Data Flow Diagram.....	23
Figure 4.1	User Interface Dependency Diagram.....	26
Figure 4.2	Tool Interface Dependency Diagram.....	30
Figure 4.3	Idraw Dependency Diagram	34
Figure 4.4	Selection Hierarchy.....	37
Figure 4.5	StateView Hierarchy.....	39
Figure 4.6	IdrawTool Hierarchy.....	39
Figure 4.7	Modified IdrawTool Hierarchy.....	42
Figure 4.8	New IdrawTool Hierarchy	49
Figure 5.1	CAPS main menu.....	54
Figure 5.2	Graphic Editor.....	55
Figure 5.3	Prototype Selector	56

ACKNOWLEDGMENTS

I have witnessed through God's work that

*"...God causes all things to work together for good to those who love God,
to those who are called according to His purpose." Romans 8:28*

This research would not have been done without His Spirit, thanks be to God.

My loving and devoted husband has encouraged me from the beginning. He gave me the encouragement to push on when the going got tough. My young daughter has inspired me to continue my education for her sake. I will always be thankful for both of them.

Lastly, I would like to thank Mac McCoy, Trish Harcum, and my chain of command: Martha Moore, Don Edwards, Jim Dooley, Rod Schmidt, and Tom Clare for supporting me throughout these past fifteen months.

I. INTRODUCTION

Increased complexity of software has led to the automated support of the software process. Such software development environments are a collection of tools used to improve both productivity and software quality. One such environment, the Computer Aided Prototyping System (CAPS), combines several tools to provide automated support for rapidly prototyping large real-time systems. CAPS constructs a prototype from specifications written in the Prototyping System Description Language (PSDL) and generates an executable model of the proposed system[Ref. 1:p. 1]. A user interface must be developed to provide interaction between the designer of the proposed system and the tools. A tool interface must be created to provide interaction between the tools. A graphic editor must be constructed to provide the designer with a means of inputting the prototype's specification graphically, without being tied to a programming language. The development of the user interface, tool interface, and graphic editor is the topic of this thesis.

A. PROBLEM STATEMENT

In order to provide a rapid prototyping environment, CAPS must support the designer throughout the entire prototyping life cycle. Although the existing tools of this environment provide the needed functionality, a user interface must be constructed which embodies the prototyping methodology and models the designer's decision process.

CAPS contains several tools that must work together to produce the final product. A tool interface must be developed to give these tools a means of interaction. It must work with the user interface to execute each tool at the proper time in response to the designer's

requests. It must also provide each tool with the proper input. The design of the tool interface must also allow for the insertion of new tools in the environment.

A rapid prototyping environment must provide the designer with a way of entering the specification of the prototype. A conceptual view of the prototype will provide the simplest means of specifying the prototype and is most easily manipulated graphically. It frees the designer from being tied to a particular programming language.[Ref. 2:p. 59] However, the other tools in the environment cannot use this graphic representation. They need to be given the prototype in a formal language. This implies that a graphic editor must give the designer the ability to input a graphical representation of the prototype, and automatically produce the formal language equivalent to the graphical input.

B. SCOPE

The development of a user interface, tool interface, and graphic editor for CAPS is the focus of this thesis. The user interface and tool interface will be used to integrate all of the CAPS tools to provide a user friendly and efficient environment. The graphic editor will let the designer input the prototype graphically.

C. RESEARCH APPROACH

Because of its ease of use, a window oriented user interface has been developed for CAPS. X Windows [Ref. 3] (or simply X), a windowing system built at MIT, has been used as the basis of the interface due to its portability and powerful applications. A toolkit has been used to develop this user interface. InterViews [Ref. 4], the chosen toolkit, provides reusable interactive components with which to build the user interface systematically. It places graphic objects on the screen without having to manipulate the windowing system directly because the X function calls are embedded in the InterViews library. Since InterViews is a C++ [Ref. 5] library, the user interface has also been written in C++.

The tool interface has been used to integrate the tool set. Communication between the tools has been carried out through this interface, and this communication is uniform throughout. The user interface interacts with the tool interface to determine when to execute a tool and what inputs to provide. The tool interface is separated from the user interface in order to provide future developers of CAPS the ability to change the user interface without changing the tool interface and vice versa.

CAPS uses the Prototyping System Description Language (PSDL) [Ref. 6] to construct the prototype. A data flow diagram (DFD) syntax with timing and control constraints has been defined for PSDL. The graphic editor has been developed to allow a designer to input this type of DFD.

A drawing editor (Idraw [Ref. 7]) provided with InterViews has been modified to produce the graphic editor. Its modification provides an internal semantic representation of the enhanced data flow diagram. The editor must provide only those functions that are needed to draw and maintain an enhanced DFD. This means some functions of the editor has been removed and others have been added. Most importantly, the editor provides multiple views of the prototype represented by the DFD. This means having the ability to open other windows while in the editor to provide other needed information about the prototype that cannot be gained from the DFD alone.

One use of multiple views includes the ability to add PSDL text while in the graphic editor. The graphic representation is insufficient to fully describe the prototype. So additional PSDL has been added. While in the editor, the designer will add this by opening another window which contains a syntax directed editor. Complete PSDL is output from the graphic editor to be used by other tools.

D. ORGANIZATION

Chapter II provides a detailed description of CAPS. The user interface toolkit will also be discussed. Chapter III describes the specification of the user interface, tool interface, and graphic editor to be built for CAPS. Chapter IV gives the architectural design of these tools. Chapter V describes the use of these tools. Chapter VI provides recommendations for further work in this area. Appendix A provides the essential model of the user interface and tool interface. Appendix B describes the PSDL grammar. Appendices C and D provides the programmers manuals for the user interface, tool interface, and graphic editor.

II. BACKGROUND

A. DEVELOPMENT OF LARGE REAL-TIME SYSTEMS

Traditionally, software has been developed using the classical project life cycle shown in Figure 2.1. The two major problems of with this type of software development is that each step cannot be started till its predecessor is completed and that implementation is mainly bottom-up[Ref. 8:p. 80]. It is not until the testing phase that it is proven that the system meets the stated requirements and specifications[Ref. 1:p. 3]. Large real time systems and systems which have hard real time constraints are not well supported by traditional software development methods. The designer of this type of system will not know if the system can be built with the timing and control constraints required until much time and effort has been spent on the implementation. A hard real time constraint is a bound on the response of a process which must be satisfied under all operating conditions.

The prototyping method shown in Figure 2.2 has recently become popular. "It is a method for extracting, presenting, and refining a user's needs by building a working model of the ultimate system - quickly and in context[Ref. 9:p. x]." This approach captures an initial set of needs and implements quickly those needs with the stated intent of iteratively expanding and refining them as the user's and designer's understanding of the system grows. The prototype is only to be used to model the system's requirements; it is not to be used as an operational system[Ref. 8:p. 95].

To manually construct the prototype still takes too much time and can introduce many errors. Also, it may not accurately reflect the timing constraints placed on the system. What is needed is an automated way to rapidly prototype a hard real time system which reflects those constraints and requires minimal development time. Such a system should make use of reusable components and validate timing constraints.

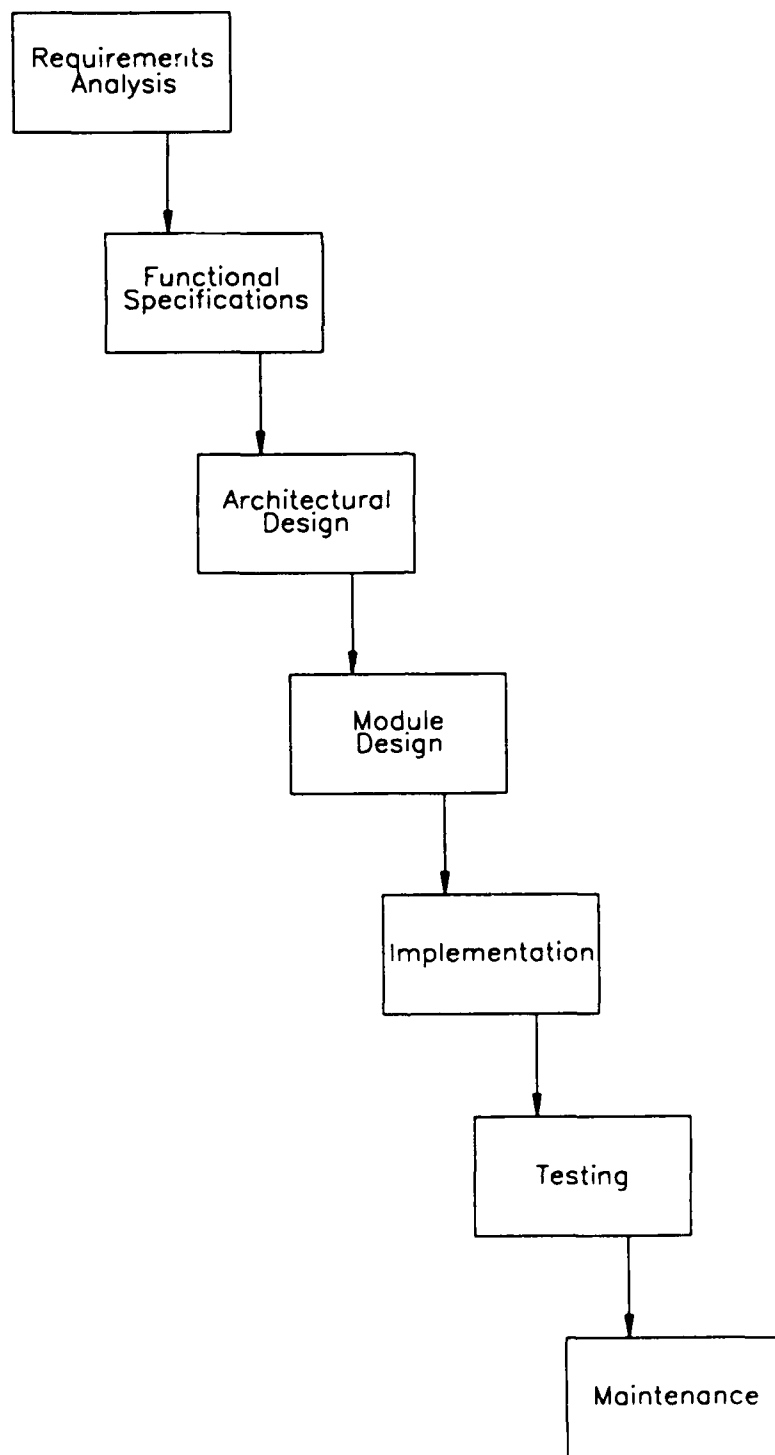


Figure 2.1: Classical Project Life Cycle [Ref. 8:p. 83]

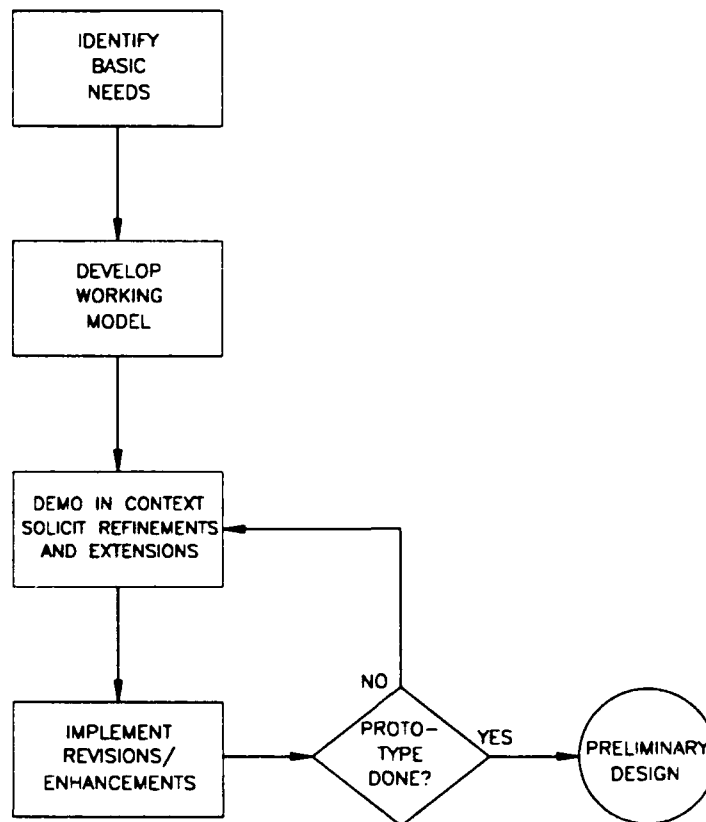


Figure 2.2: Prototyping Life Cycle [Ref. 9:p. 8]

B. THE COMPUTER AIDED PROTOTYPING SYSTEM

The Computer Aided Prototyping System (CAPS) is a set of software tools. It provides an environment for rapidly constructing executable prototypes. The designer of a software system uses a graphic editor to create a graphic representation of the proposed system. The graphic representation is used to generate part of an executable description of the proposed system, represented in the prototyping language. This description is used to search a database of reusable software components to find components to match the specification of the prototype. A transformation schema is used to transform the prototype into a programming language. The prototype is then compiled and executed. The end user of the proposed system will evaluate the prototype's behavior against the expected behavior. If the comparison

results are not satisfactory, the designer will modify the prototype and the user will evaluate the prototype again. This process will continue until the user agrees that the prototype meets the requirements.

The CAPS addressed in this thesis is based on the Prototyping System Description Language (PSDL). "It was designed to serve as an executable prototyping language at the specification or design level[Ref. 10:p. 26]." It can be expressed graphically, annotated with text.

PSDL is based on the following mathematical model:

$$G = (V, E, T(V), C(V))$$

where V is the set of vertices, E is the set of edges, $T(V)$ is the set of timing constraints placed on V , and $C(V)$ is the set of control constraints placed on V . A vertex represents an operator which is a function or state machine. An edge represents a data stream which carries a value of an abstract data type. A timing constraint is the maximum execution time of an operator. "Control constraints are used to limit an operator's behavior by specifying conditions regarding its firing (execution) or I/O processing." [Ref. 11:p. 3]

1. Prototype Construction Using CAPS

The CAPS is composed of eight tools: graphic editor, syntax directed editor, software base, design database, translator, static scheduler, dynamic scheduler, and an Ada compiler. These tools are used together to rapidly construct the prototype. The steps used in the construction process are described below.

a. Edit

The designer uses the graphic editor to draw an enhanced data flow diagram (DFD) that represents the proposed system. A DFD contains data flows (directed lines) and operators (circles). The operators represent functions or state machines, and the data flows represent data used as input or output to the functions of state machines. The DFD is enhanced with the timing and control constraints needed by the system.

The current implementation of the graphic editor produces a partial PSDL representation of the prototype containing the links between the DFD components. A detailed description of the new implementation of the graphic editor is given later in this thesis.

The syntax directed editor is used to add extra PSDL information about each operator and abstract data type that cannot be determined from the drawing alone. The syntax directed editor is a language-based editor tailored to PSDL. It assists the user in creating a syntactically correct description of the prototype written in PSDL. It provides templates containing the legal alternatives for PSDL constructs. It combines text editing with incremental parsing techniques to guarantee that the result is syntactically correct[Ref. 12:p. 57].

In the previous implementation of CAPS, the syntax directed editor is used as a separate tool and is called from the user interface. The problem with this approach is that when the DFD is changed in the graphic editor, the PSDL must be regenerated using the syntax directed editor. Inconsistencies between the DFD and the PSDL can occur. A detailed description of how the syntax directed editor is integrated with the new implementation of the graphic editor is given later in this thesis.

b. Search For Reusable Components

The PSDL specification of an operator can be used to find reusable components stored in the software base, a database of reusable components. If a reusable component is not found, the designer must decompose the operator using the graphic editor, or he must implement a component directly in Ada. If decomposition is performed, the software base can be searched for reusable components that match this decomposition. This method of searching and decomposition is repeated until the desired reusable components are found. Work is currently in progress on the development of the software base.

c. Translate

The PSDL generated during the editing process is used to generate Ada code which has the timing and control constraints built in. The translator translates the PSDL

specification of the prototype into Ada code. It constructs an abstract syntax tree by analyzing the PSDL representation of the prototype[Ref. 12:p. 102]. This tool also binds the reusable components found in the software base and any components written by the designer to the executable prototype.

The static scheduler uses six scheduling algorithms to link together all of the operators with timing constraints into an executable schedule that guarantees all of the timing constraints are met. Currently, three of the six algorithms have been integrated into CAPS.

The dynamic scheduler adds the operators without timing constraints to the executable schedule produced by the static scheduler. It uses the time slots that are not used by the time critical operators[Ref. 1:p. 8].

d. Compile and Execute

CAPS uses an Ada compiler to compile and link together the reusable components, any components specific to the prototype, and the code produced by the translator, static scheduler, and dynamic scheduler. It produces an executable version of the prototype which can then be executed inside of CAPS.

e. Store Prototype Representations

Each of the tools produce a different representation of the prototype. The design database is used to store each of these representations of the current prototypes. The output of each tool is stored in the design database after the tool has been executed. CAPS retrieves the prototype in the form required as input to each of the CAPS tools.

Work is currently in progress on the development of the design database. A pseudo database currently exists as a defined directory structure and a shell script for accessing files from that structure. See Appendix C, page 101 for the implementation of this pseudo database.

C. INTERVIEWS: A USER INTERFACE TOOLKIT

InterViews is a C++ graphical interface toolkit developed at Stanford University [Ref. 13]. It consists of a library of predefined interactive objects. These objects can be combined to construct many types of user interfaces[Ref. 4:p. 3]. InterViews' object oriented approach provides a natural way to build user interfaces as communicating objects.

InterViews runs on top of X Windows, a windowing system developed at MIT. X is a portable windowing system that can be run on a variety of workstations. "It offers a rich and complex environment to the programmer and user of application software [Ref. 14:p. 2]." The X environment is built upon the base window system. The X network protocol provides interaction between the outside world and the base window system. X provides a low level programming interface (Xlib) to the network protocol. High level toolkits, like InterViews, have been developed to mask some of the complexity of Xlib and the network protocol[Ref. 3:p. 3].

InterViews has several strengths:

- InterViews' object oriented design provides a simple organization of user interface classes that is simple to use and extend via subclasses.
- InterViews supports consistency across applications[Ref. 15:p. 20].
- InterViews allows customization of the user interface by the end user. It uses X Toolkit's version of the resources to do this.
- InterViews abstracts X from the user interface. Because of this, the user interface code does not have to refer to low level details of X.
- InterViews separates the user interface from the application. This will provide us with the opportunity to change the user interface without changing the underlying functionality.

InterViews provides three classes of objects: interactors, graphics, and text. (1) "An interactor class manages some area of potential input and output on a workstation display[Ref. 4:p. 3]." All user interface objects are inherited from this class. Examples of interactors are buttons, menus, and dialogue boxes. Figure 2.3 shows the hierarchial structure of the interactor class. (2) The graphics class defines structured graphics objects. Objects

in this class can draw and erase themselves. A state is attached to each object which includes attributes of color, line style, and coordinate transformation[Ref. 13:p. 7]. Examples of graphics include circles, ellipses, and lines. Figure 2.4 shows the hierarchial structure of the graphics class. (3) The text class defines structured text objects like strings, text editors, and text buffers.

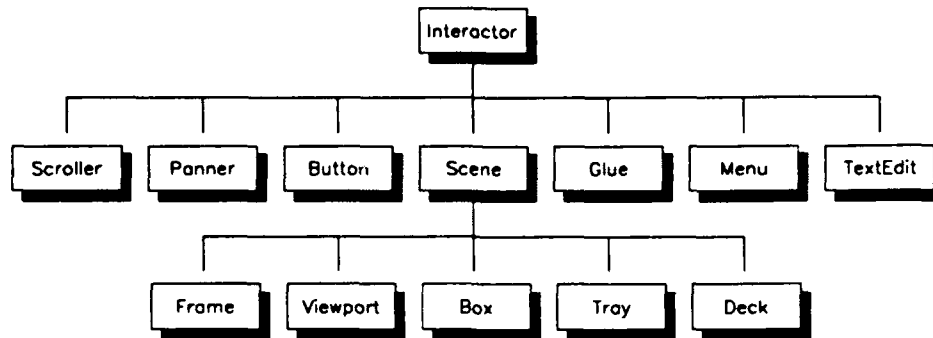


Figure 2.3: Interactor Class Hierarchy [Ref. 13:p. 2]

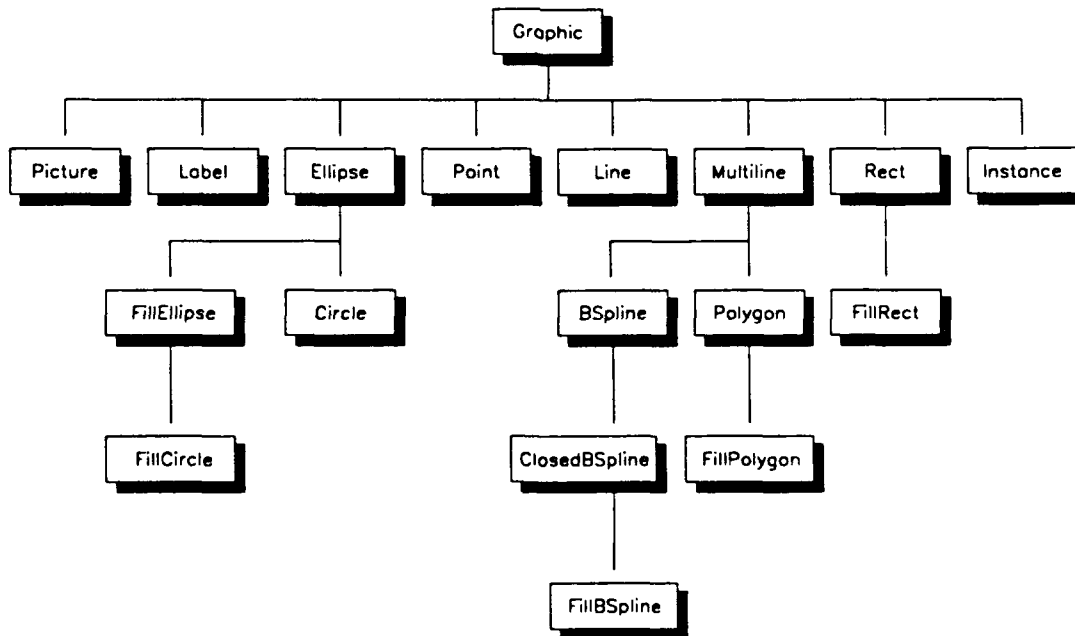


Figure 2.4: Graphics Class Hierarchy [Ref. 7:p. 3]

D. IDRAW: INTERVIEWS DRAWING EDITOR

Idraw is an object oriented drawing editor provided with InterViews. Idraw provides immediate feedback as the user creates and manipulates graphic objects such as circles, lines, and splines. An example of this type of feedback is the movement of a rectangle causes its outline to follow the mouse. A picture of Idraw is shown in Figure 2.5. Its user interface is composed of InterViews' interactor and graphics classes. It contains a set of tools on the left side of the editor and a set of pull down menus along the top. The tools are used to create the graphic objects or to change the user's view of the drawing. The menus are used to set or change a graphic object's attributes.

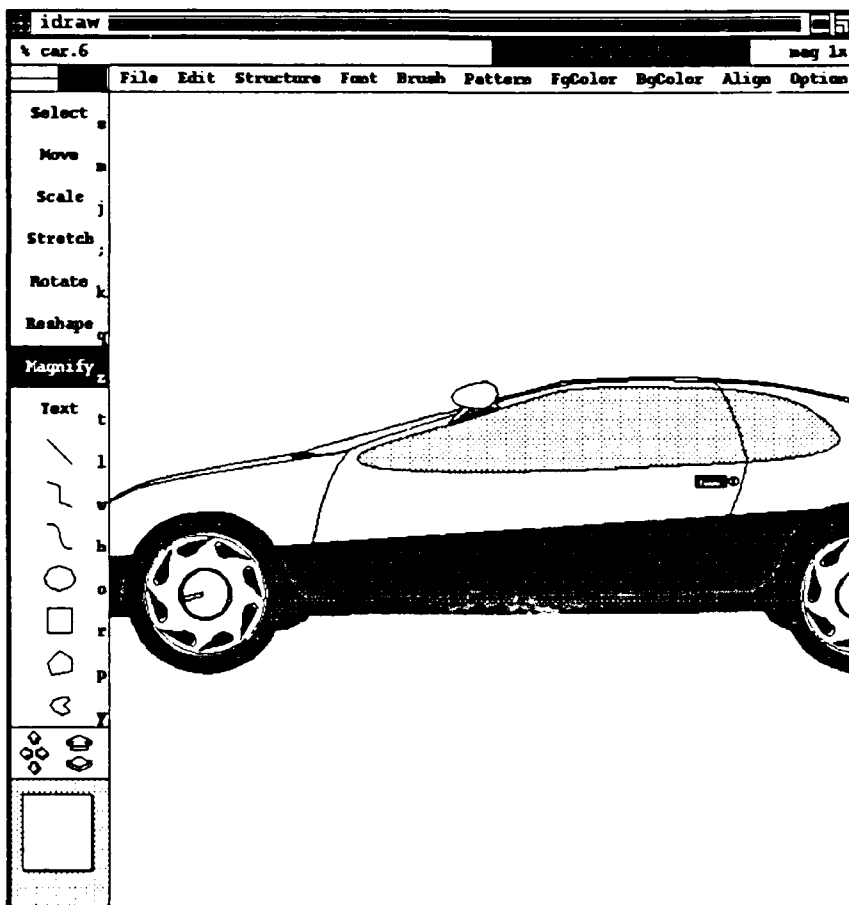


Figure 2.5: Idraw: InterViews Drawing Editor

Idraw allows the user to create the following objects:

- line
- multi line
- open spline
- ellipse
- rectangle
- closed spline
- polygon
- text.

Once drawn, these objects can be moved, scaled, stretched, rotated, or reshaped. Any part of the drawing can be magnified. Various standard editing functions, such as printing a drawing, saving a drawing, and deleting an object, are provided. Certain attributes of the editor, such as font types, brush type, pattern, and foreground and background color can be customized by setting X resources. These customizations are stored with the drawing so they can be restored when the drawing is edited. The drawings are stored in a modified PostScript format.

III. REQUIREMENTS SPECIFICATION

This chapter presents a summary of the requirements for the user interface, tool interface, and graphic editor. The Yourdon [Ref. 8] method of requirements analysis is used to develop a model of what these systems must do in order to satisfy the requirements. This model is called the essential model and is broken into two components: the environmental model and the behavioral model. The environmental model describes the interface between the system being modelled and the other systems and tools it must interact with. The behavioral model describes the internal processes of the system.[Ref. 8:p. 323,324]

The term *CAPS interface* will be used to represent the *user interface* and *tool interface* together as one unit. Each of these will be subsystems of the *CAPS interface*.

A. CAPS INTERFACE

The CAPS interface provides a cohesive software development environment integrating the tools of CAPS. A pictorial representation of this environment is given in Figure 3.1. At the core of the environment is the host operating system. The windowing system, X-windows, is the next layer. The toolkit chosen to develop the user interface, InterViews, provides the interaction between the upper layers of the environment and X. The CAPS tools sit on top of InterViews and are surrounded by the tool interface. The tool interface provides all communication between the tools and the user interface. The user interface, the outer most layer of the environment, is what the designer sees and hides the underlying implementation details and interfaces from the user.

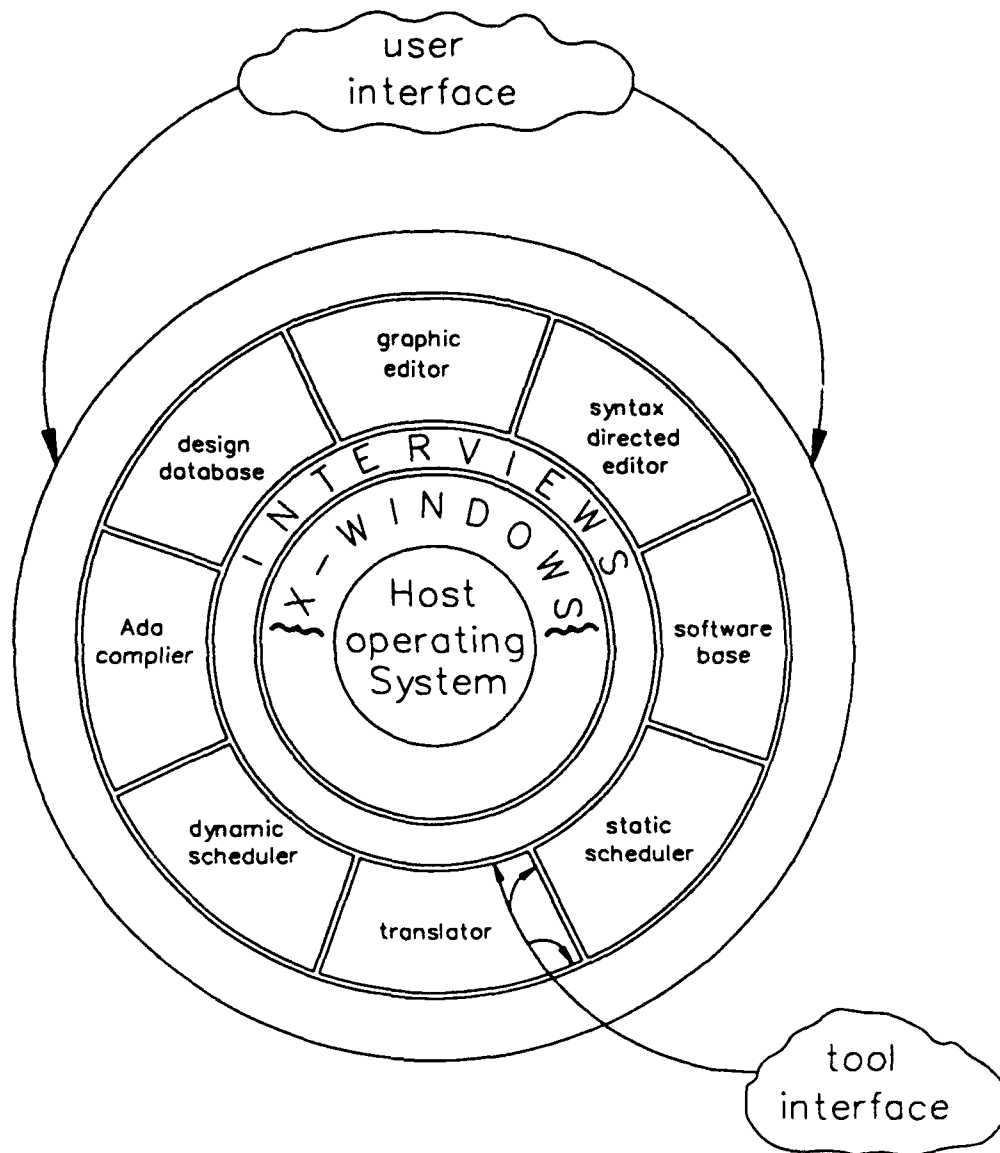


Figure 3.1: The CAPS Environment

One point deserves clarification here. If a tool needs to interact with the designer during its execution, it is assumed that it will provide its own user interface. The CAPS user interface is not responsible for providing the interaction between a particular tool and the designer while the tool is running.

1. The Environmental Model

Appendix A contains the environmental model in its entirety including the context diagram, event list, and statement of purpose. A summary of the model is given below.

The CAPS interface shall communicate with a number of external entities. These entities include the designer and the tools of CAPS. The data passed between the tools, the designer, and the CAPS interface is shown in Figure 3.2.

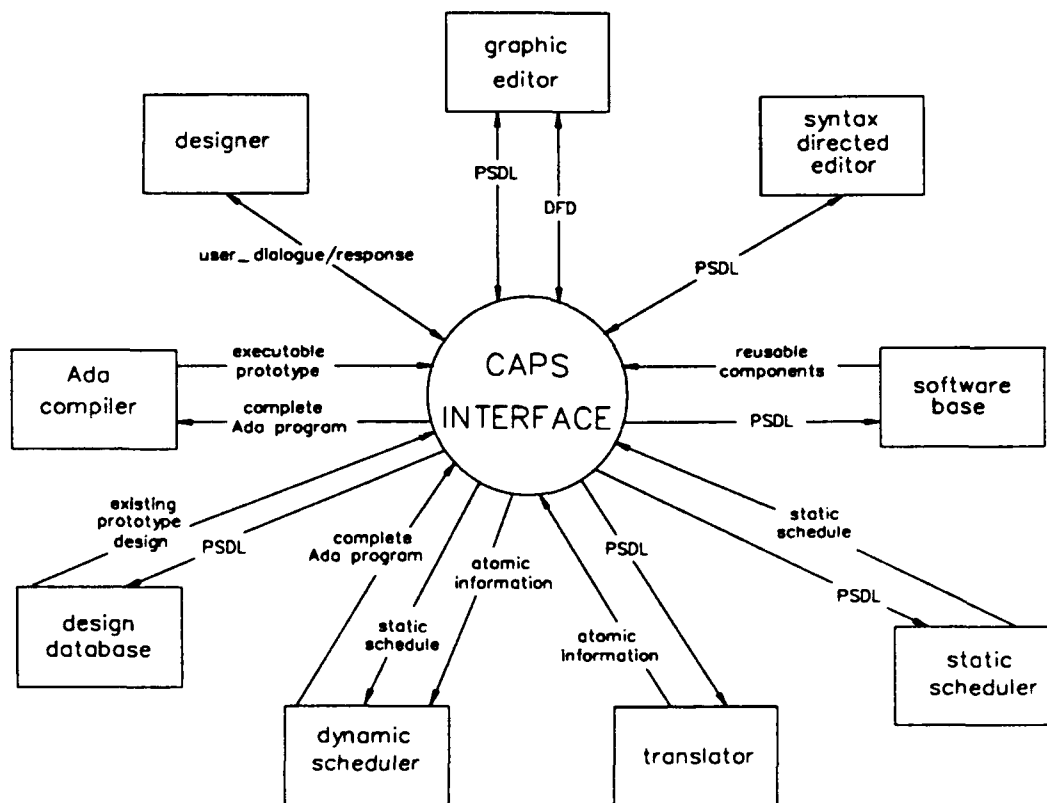


Figure 3.2: CAPS Interface Context Diagram

The graphic editor shall require a DFD drawing and the PSDL as input. The modified drawing and PSDL will be output from the graphic editor. The syntax directed editor shall modify the file of PSDL from the graphic editor. The software base shall use the PSDL file to create the reusable components that match the specifications listed in the PSDL file. The translator shall use the PSDL file to create the atomic description of the operators. The static scheduler shall use the PSDL file to create the executable schedule of the operators with timing constraints. The dynamic scheduler shall use the output of the translator and static scheduler to create the complete Ada representation of the prototype. The compiler shall use that Ada code to create an executable prototype.

The CAPS interface must respond to events that occur during its execution. Events are caused by external entities and are given in the event list in Appendix A.

2. The Behavioral Model

Appendix A contains the environment model in its entirety including the data flow diagrams, data dictionary, and the process specifications. A summary of the model is given below.

The CAPS interface is divided into two subsystems, the *user interface* and the *tool interface*. Figure 3.3 shows this decomposition. The user interface shall provide an abstraction of the CAPS tools to the designer. It gives the designer a consistent way to invoke each tool and select a prototype for use by each tool. The tool interface shall provide for communication among CAPS tools and between tools and the user interface. The type of communication between the two interfaces shall be through the passing of the prototype name and the function that the designer wants to run.

The user interface does not provide direct access to all the CAPS tools; the tools are grouped to perform basic functions of the rapid prototyping life cycle. The designer

will choose a particular function to execute from the user interface. The basic prototyping functions are:

- edit/create
- search
- translate
- compile
- execute.

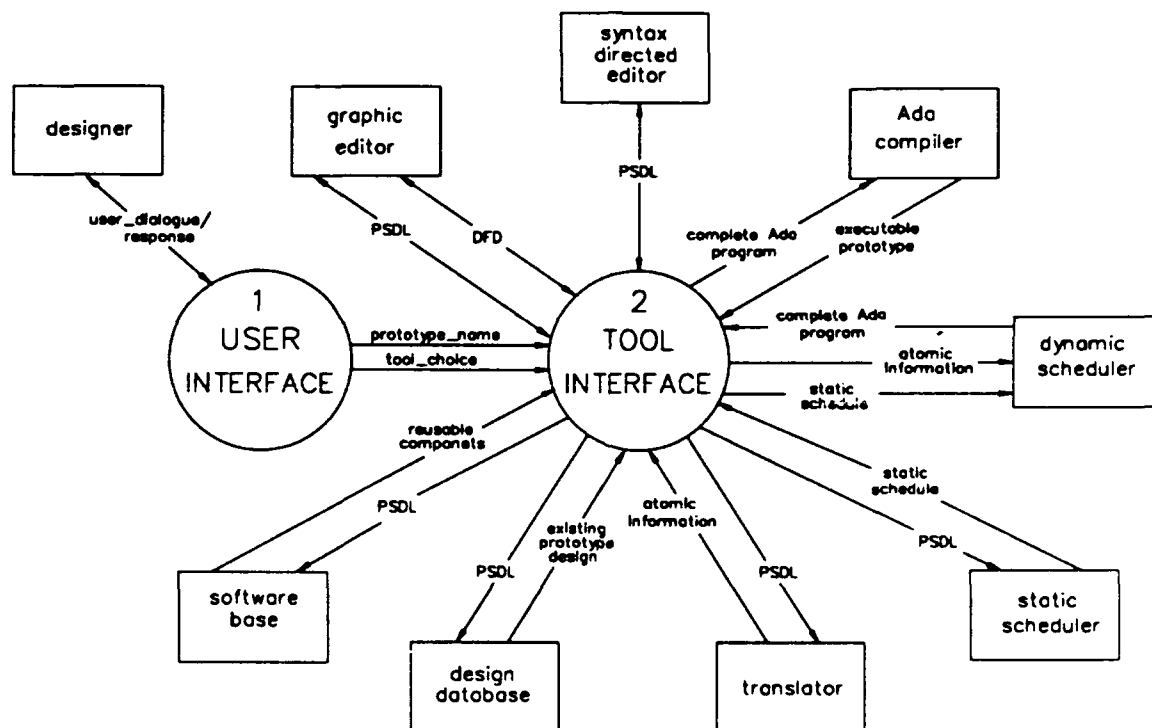


Figure 3.3: CAPS Interface Data Flow Diagram

The prototyping life cycle begins by the designer drawing or editing a data flow diagram using the graphic editor producing PSDL code. Then, the designer uses the syntax directed editor to create specifications for the operators and types. The designer searches the software base for reusable components to match the PSDL specification of each of the operators. The PSDL is translated into Ada. The Ada code is compiled to make an executable prototype. The last step is to execute the prototype. If an error is found or the prototype is unacceptable at any of these stages, the designer can redo any or all of the steps.

The user interface shall present the functions to the designer and wait for a selection. Once the selection is made, the user interface queries the designer for the name of the prototype to use when performing the function. Figure 3.4 presents the data flow diagram of this process.

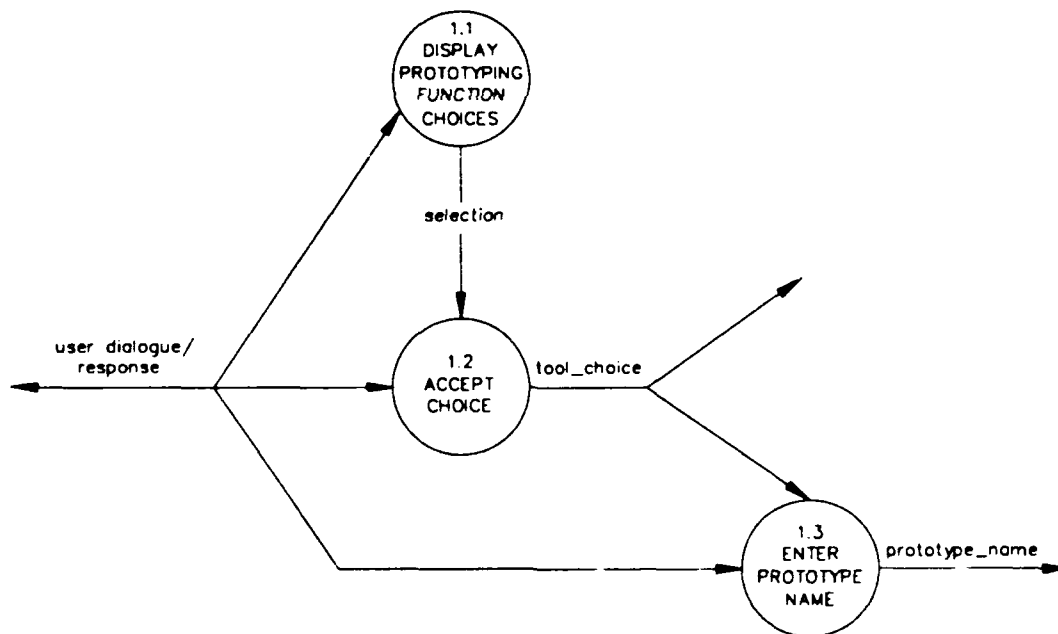


Figure 3.4: User Interface Data Flow Diagram

The tool interface shall use the tool and prototype selection to determine what tools to run, what order to run the tools, the input to send to each tool and the output to receive from each tool. It shall handle the concurrent execution of the tools.

3. Principles of User Interface Design

In addition to the requirements summarized above, eight general principles of user interface design shall apply to the development of the user interface.[Ref. 16:p. 60-62]

- Provide a consistent means of interaction throughout the program.
- Provide informative feedback to the user.
- Organize the dialogue into a beginning, middle, and end.
- Enable the experienced user to use shortcuts.
- Offer simple error handling.
- Permit easy reversal of actions.
- Make the user feel in control.
- Reduce the user's short term memory load.

B. GRAPHIC EDITOR

The graphic editor shall support the creation and modification of the graphical representation of PSDL prototypes. [Ref. 12:p. 36] Since prototypes may be multi level, the editor shall provide a mechanism for decomposing the components of the drawing into subcomponents. The editor shall automatically generate a PSDL representation which captures the meaning intended by the drawing. [Ref. 10:p. 22] A summary of the essential model of the Graphic editor is given below.

1. The Environmental Model

The graphic editor shall communicate with the tool interface and the designer. The data passed between these external entities and the editor is shown in Figure 3.5. The editor

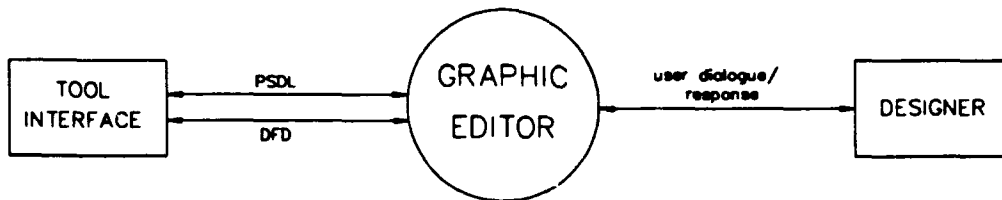


Figure 3.5: Graphic Editor Context Diagram

receives the file containing the PSDL representation of the prototype and the drawing file from the tool interface. These files are modified during the editing session and returned to the tool interface when the editor is terminated. The editor shall interact with the designer through dialogue and response.

2. The Behavioral Model

The graphic editor is divided into five subsystems as shown in Figure 3.6. The editor shall use the drawing file and PSDL file to initialize the editor's drawing and PSDL representation of the drawing. The editor then displays the window to the designer, waits for the designer's editing choice, and processes the choice. This is done until the editor is terminated.

The window displayed to the user shall contain a list of drawing tools. The following is a minimum list of tools to be provided:

- Draw a circle to represent an operator.
- Draw a straight or curved line to represent a data flow.
- Draw a curved line to represent a self loop.
- Add comments to the drawing. Add labels to the drawn objects.
- Decompose an operator into suboperators.
- Move a part of the drawing to another location on the display.
- Edit PSDL text which describes an object. The editor will automatically generate a portion of the PSDL from the drawing. The designer will edit this text to complete the PSDL.

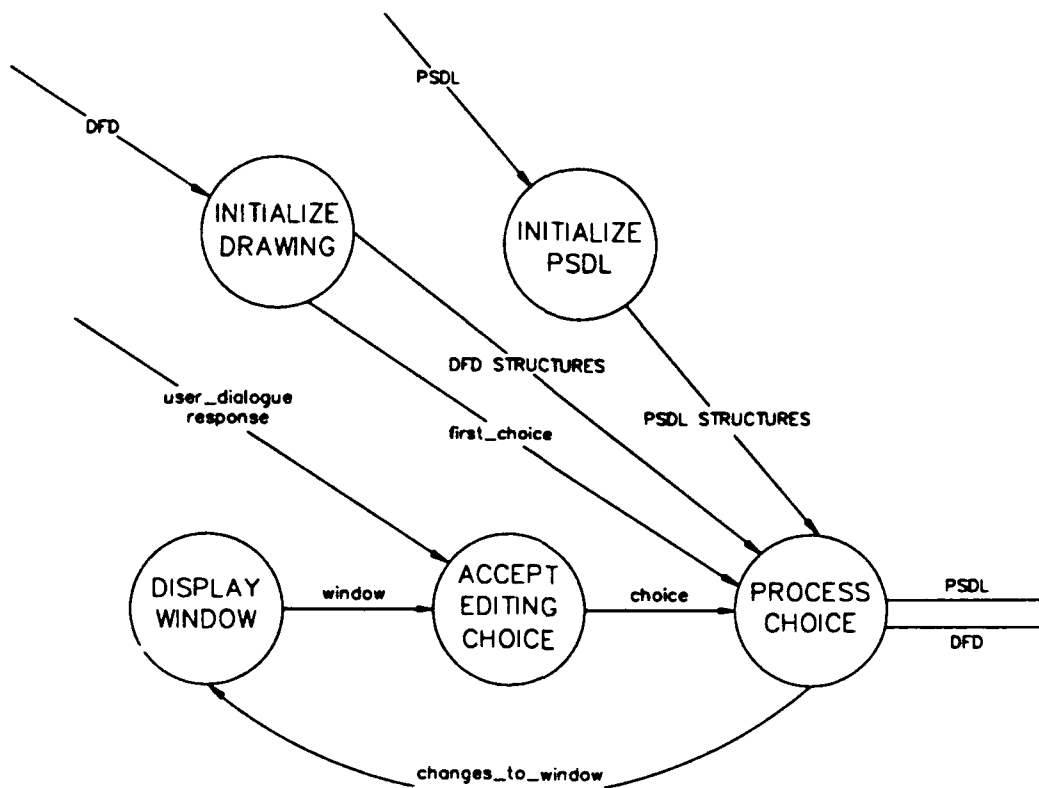


Figure 3.6: Graphic Editor Data Flow Diagram

In addition to the above tools, the graphic editor shall present a list of common editing functions to choose from. A minimum set of functions include:

- Open a drawing.
- Save a drawing.
- Print the drawing.
- Delete a DFD component.

IV. ARCHITECTURAL DESIGN

In this chapter, the architectural design of the user interface, tool interface, and graphic editor is described. An object oriented approach is used to create a representation of these systems in terms of the entities that exist in the problem space. The user interface, tool interface, and graphic editor are modeled in terms of their *classes*, their *attributes*, and the *operations* they perform. The requirements given in Chapter III are used as a basis for modeling these systems.

The object oriented design methodology consists of the following three steps:

- Develop informal solution.
- Develop formal solution by identifying attributes and operations.
- Develop implementation strategy.

An informal solution is composed of a single paragraph, describing the behavior of the system. The verbs used in the paragraph are mapped to operations in the formal solution. The nouns are mapped to objects or classes.

A formal solution describes the classes and external systems which define the system. This information is presented using a dependency diagram and a class hierarchy. A dependency diagram consists of boxes and directed lines. Boxes represent external systems or classes. Directed lines establish an interface between classes and/or external systems. A class hierarchy lists the classes and external systems which are components of the system and describes their behaviors, attributes, and operations.

Many of the terms used in the class hierarchy are described here. A *class* is a user-defined data type. It refers to a set of data elements and operations that act on that data.

Objects are instances of a class. An object can also be an external system. Each of the data elements in a class is an *attribute* of the class. In C++, these are known as *member variables*. The operations on this data are known as *member functions* in C++. All classes have a *constructor* operation to perform needed initialization when an instance of the class is created. The constructor has the same name as the class. A colon (:) will be used to relate an object to its class. For example, `circle : Circle` shows that `circle` is an instance of the class `Circle`.

An implementation strategy provides a basis for implementing the system in terms of the target X Windows toolkit, InterViews. It describes which InterViews classes might best be used in the implementation of the class.

A. USER INTERFACE

1. Informal Solution

The user interface displays a menu of prototyping functions to the user and waits for the user to choose an item. Once chosen, a list of prototypes available for use by that function are displayed. After the user selects one of the prototypes, the CAPS tools that correspond to the chosen function are executed.

2. Formal Solution

The major classes and their interrelationships are shown in Figure 4.1. The class hierarchy is as follows:

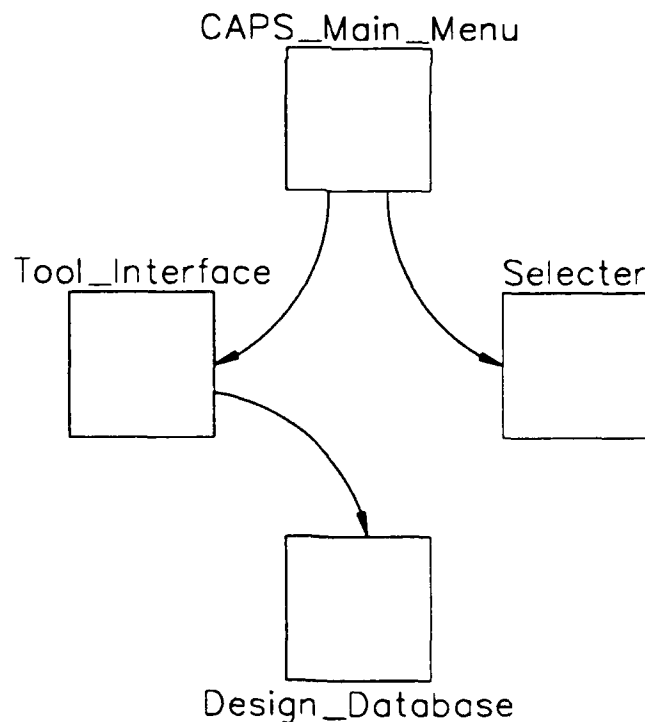


Figure 4.1: User Interface Dependency Diagram

Class CAPS_Main_Menu

Behavior: Displays a menu of CAPS prototyping functions.

Attributes:

Set_Of_Functions: Group of prototyping functions.

Operations:

CAPS_Main_Menu: Creates and displays menu.

Accept: Processes user's menu choice.

Class Selector

Behavior: Displays a menu of prototype names available for use by chosen function.

Attributes:

List_Of_Prototypes: List of prototypes for chosen function.

Operations:

Selector: Creates and displays menu.

Insert: Inserts available prototypes into menu.

Select: Processes user's prototype choice.

System Tool_Interface

Behavior: Manages the execution of the CAPS tools.

Attributes:

Function_Choice: Function chosen in CAPS_Main_Menu.

Prototype_Name: Prototype chosen in Selector.

Operations:

Tool_Interface: Initializes all attributes.

ExecuteFunction: Executes CAPS tool(s) that correspond to chosen function.

System Design_Database

Behavior: Manages files associated with a prototype.

Attributes:

Function_Choice: Function chosen in CAPS_Main_Menu.

List_Of_Prototypes: List of available prototypes for chosen function.

Operations:

Design_Database: Initializes all attributes.

FindPrototypes: Locates all available prototypes for chosen function.

3. Implementation Strategy

Operation Main

The user interface should be driven by the Main program. I should create a world for the user interface using the InterViews class **World**. It should create an instance of the **CAPS_Main_Menu** class and insert the menu into the world. It should then use the **Accept** operation of the **CAPS_Main_Menu** to loop on the user's menu choices until the menu is terminated. Lastly, it should remove the menu from the screen.

Class CAPS_Main_Menu

The **CAPS_Main_Menu** should be a dialog box containing push buttons representing the CAPS functions. The user will choose the desired function by pressing one of the buttons using the mouse. A dialog box containing the names of the prototypes available to be used with that function should then pop up. The user can choose a name or type the name of a new prototype in the space provided. The **Tool_Interface** should execute the proper tool(s) corresponding to the function chosen.

Class Selector

The **Selector** is inherited from the **StringChooser** class, an interactor class provided with InterViews. The **StringChooser** is a dialog box which manages keyboard focus between a **StringBrowser** and a **StringEditor**. A string can be selected with the mouse in the **StringBrowser** or can be typed in the **StringEditor**. The **Selector** builds upon the **StringChooser** to present prototype names to the designer. Two buttons are provided: **Select** and **Cancel**. The **Select** button is pressed once a name is chosen in order to process the chosen name, and the **Cancel** button is used to terminate the selection without executing the CAPS function.

System Design_Database

The **Design_Database** will be used to retrieve the names of the available prototypes. Since it has not been implemented, a simulated design database will need to be written to search a predefined directory for all files which can be input to the chosen function.

System Tool_Interface

The **Tool_Interface** is described in detail in Section B.

B. TOOL INTERFACE

1. Informal Solution

The tool interface executes a CAPS tool based on the prototyping function requested. It uses the prototype selected to determine the input and output for the tool.

2. Formal Solution

The tool interface depends on the CAPS tools. A diagram of these dependencies is shown in Figure 4.2. The class hierarchy follows.

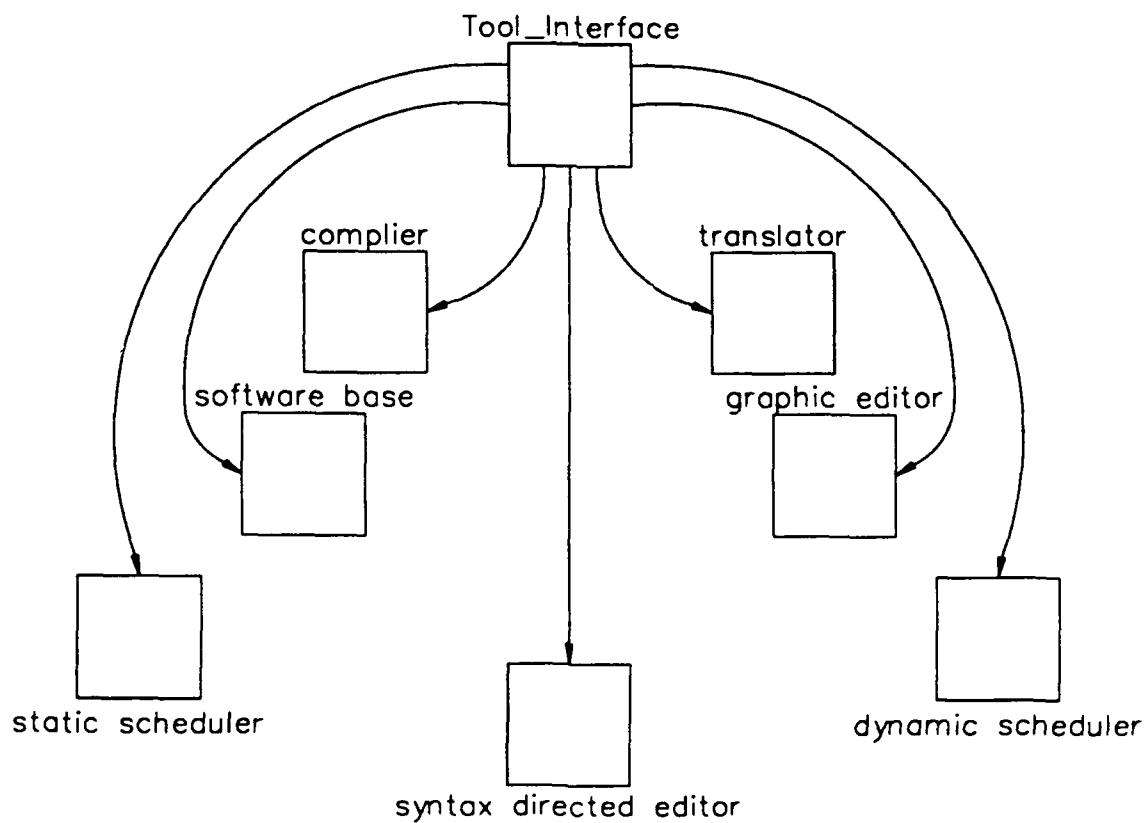


Figure 4.2: Tool Interface Dependency Diagram

Class Tool_Interface

Behavior: Executes the CAPS tool(s) based on prototyping function requested.

Attributes:

Function_Requested: Request , in the form of a query, made by user interface or any of the CAPS tools.

Prototype_Name: Prototype chosen in user interface.

Operations:

Tool_Interface: Initializes class variables and parses input parameters.

ExecuteFunction: Retrieves appropriate inputs based on prototype name and executes function using CAPS tool(s).

System graphic editor:

Behavior: Used to create an enhanced DFD and the PSDL that represents it. It takes as input the existing drawing and the PSDL (if already created).

System syntax directed editor:

Behavior: Used by the graphic editor to add extra PSDL information about an operator or data type. It takes as input the partial PSDL created from the drawing.

System software base:

Behavior: Used to find reusable components to match the operators drawing in the graphic editor. It takes as input the PSDL specification for each operator.

System translator:

Behavior: Used to translate the specification of the operators into Ada. It takes as input the PSDL created in the graphic editor.

System static scheduler:

Behavior: Used to schedule the operators with timing constraints. It takes as input the PSDL created in the graphic editor.

System dynamic scheduler:

Behavior: Used to schedule the operators without timing constraints. It takes as input the PSDL created in the graphic editor.

System compiler:

Behavior: Used to compile the reusable components, and the output of the translator, the static scheduler and the dynamic scheduler.

3. Implementation Strategy

Operation Main

The tool interface is driven by the **Main** program. It will perform two functions: parse the given arguments, and execute CAPS tool(s) based on the function requested.

Class Tool_Interface

The **Tool_Interface** will use the given **function_request** and the **prototype_name** to retrieve data from the design database to pass as input to the tools. It will execute the tool(s) by opening a terminal window in X. It then gets output from the tools and updates the design database.

C. GRAPHIC EDITOR

The method chosen for construction of the graphic editor is the modification of an existing drawing tool. Two advantages of this approach are:

- The editor can be developed in less time.
- The drawing tool will provide much of the required functionality.

InterViews' Idraw was chosen as the drawing tool because of its powerful graphics, PostScript output, editing capabilities, object oriented representation, and ease of use. It also provides a variety of fonts, colors, brush types, and patterns to enhance the drawing.

An iterative design technique will be used to design the graphic editor. The first step is to analyze Idraw. After this, the following steps will be done iteratively:

- Select an area to change.
- Modify high level design to incorporate the change.
- Implement and test the change.

1. Analyze Idraw

Idraw is an InterViews application written in the C++ object oriented language. See Figure 2.5 for a display of Idraw. The following is an object oriented analysis of Idraw.

Class Idraw

Idraw is the main class of the editor. It opens a given drawing file, if any, creates the editor on the screen, and calls the event handler, **Run**, to process the user's actions until the user chooses to terminate the editor.

Idraw depends on several classes. A diagram of these dependencies is shown in Figure 4.3. The class hierarchy is as follows:

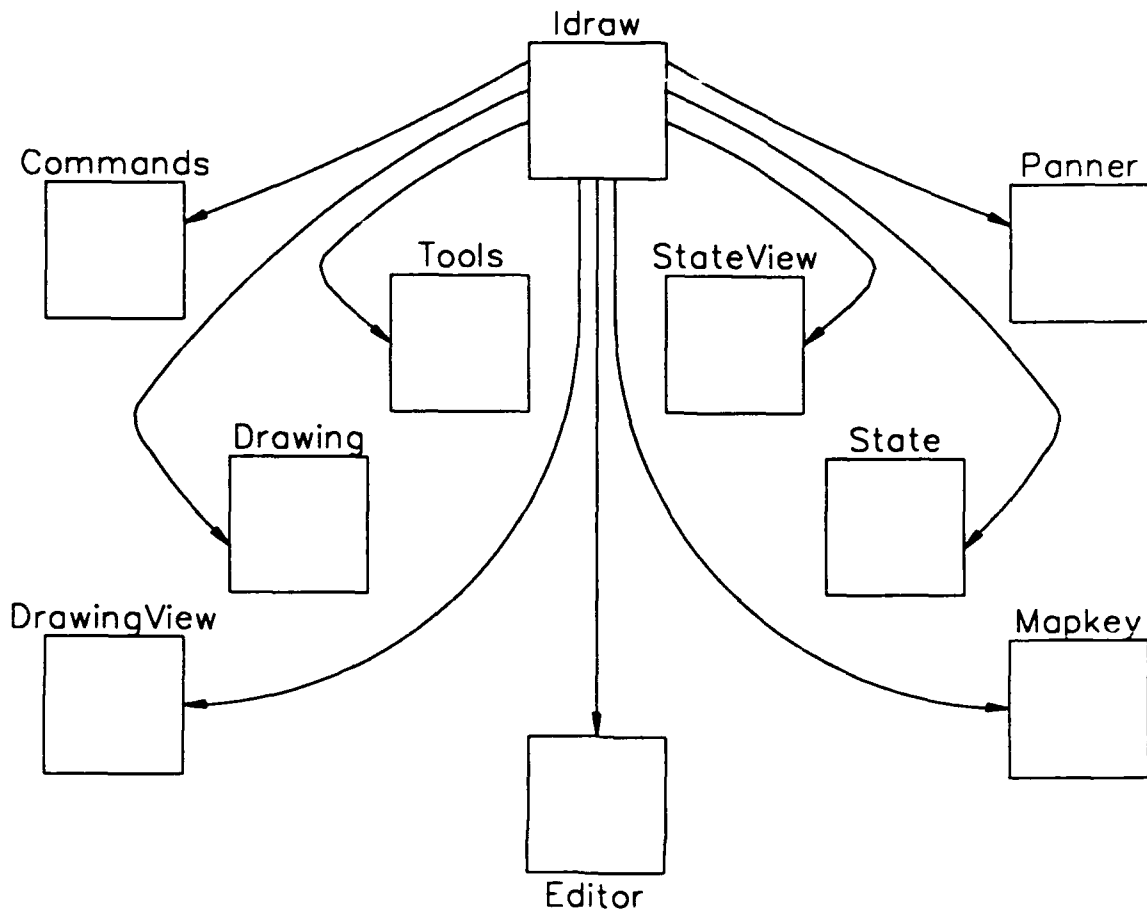


Figure 4.3: Idraw Dependency Diagram

Class Idraw

Behavior: Displays a drawing editor.

Set of Attributes:

initial_drawing:	Name of drawing file to edit.
cmds : Commands:	Displays a pull down menu bar which contains many pull down menus.
drawing : Drawing:	Performs operations on the drawing.
drawingview : DrawingView:	Displays drawing.
editor : Editor:	Handles drawing and editing operations.
mapkey : MapKey:	Maps characters to Interactors.

panner : Panner:	Pans and zooms drawing.
state : State:	Stores current state information about drawing.
stateview : StateView:	Displays current state information.
tools : Tools:	Displays drawing Tools.
<i>Operations:</i>	
Idraw:	Parses command line arguments, initializes attributes, and displays editor.
Run:	Opens drawing file, and processes user's choices until the editor is terminated.

Class Commands

Commands provides over 100 traditional editing commands. It is displayed as a menu bar containing a collection of pull down menus referenced by the title of the menu.

The following groups of editing functions are represented by the menus:

- **File:** Presents file operations such as opening a file, saving a file, and printing a file.
- **Edit:** Presents editing operations such as deleting a selected object, cutting an object from the drawing, and pasting a cut object in the drawing.
- **Structure:** Presents operations to change the way objects are structured together such as grouping selected objects together, bringing an object to the front of the group, and sending an object to the back of the group.
- **Font:** Presents a list of fonts to be used for the text in the drawing.
- **Brush:** Presents various types of brushes such as solid lines, dashed lines, and directed lines anchored at one end.
- **Pattern:** Presents a list of patterns for the selected graphic objects.
- **Color:** Presents a list of foreground and background colors for the selected graphic objects.
- **Align:** Presents various alignment options such as aligning left sides of selected objects, and aligning centers of selected objects.
- **Options:** present various options to aid in putting the drawing on the page, such as reducing a drawing to fit on one page, centering a drawing to the page, and providing a grid.

Class Drawing

The **Drawing** contains the internal representation of the picture. The user's drawing is stored as a linked list of graphic objects. The interface to modify the objects is also provided. **Drawing** depends on two classes:

- **PictSelection:** Linked list of objects in drawing.
- **SelectionList:** Linked list of those objects to be modified by a command or tool.

Class DrawingView

The **DrawingView** provides the user's view of the drawing. It is responsible for everything drawn on the screen.

Class Editor

The **Editor** performs the operation selected by the user for the given tool or command. It uses **Drawing** to modify the internal representation of the drawing when needed.

Class MapKey

Each of the tools and commands can be executed by typing a letter as a shortcut to clicking with the mouse. **MapKey** maps the letter to the tool or command desired. It stores all of the tools and commands in a table indexed by the shortcut letter.

Class Panner

The **Panner** is used to pan and zoom the drawing in order to view the drawing close up or farther away.

Class PictSelection

Drawing stores all of the graphic objects drawn on the screen in picture, an instance of the **PictSelection** class. Each object is an instance of a class inherited from the class **Selection**. **Selection** is inherited from the InterViews Graphic class. **PictSelection** is

inherited from **Selection** and is a linked list of all the **Selections** in the drawing. The **Selection** inheritance hierarchy is shown in Figure 4.4. The **NPtSelection**, also shown in the figure, is a special type of graphic object that can draw arrowheads on one or both of its endpoints.

Class SelectionList

The **SelectionList** contains a subset of the **Selections** in the drawing. It is a linked list of only those **Selections** chosen by a drawing tool or editing command.

Class State

State stores state information about the user's drawing and paint attributes to be used when creating new graphic objects. Some of the information stored is the brush type, drawing name, font, and pattern.

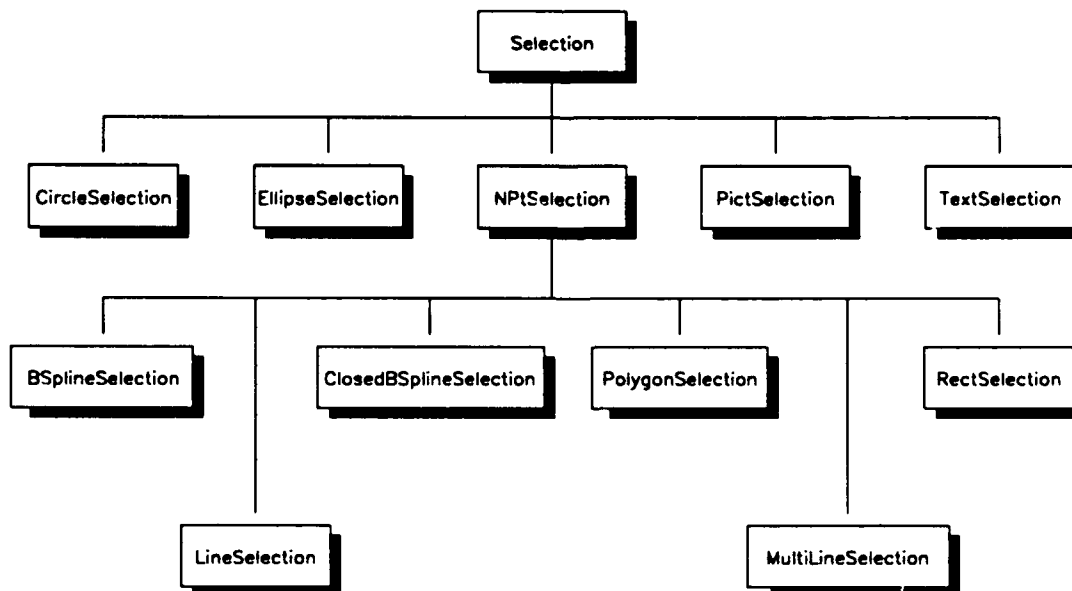


Figure 4.4: Selection Hierarchy

Class StateView

Idraw keeps track of and displays current state information. The kinds of information displayed are:

- current brush type
- name of current drawing
- current font
- status of gridding
- magnification percentage
- current modification status of drawing
- current pattern.

Each piece of information is a separate object and inherited from **StateView**. The **StateView** hierarchy is shown in Figure 4.5.

Class Tools

Tools creates the panel that displays the drawing tools to the user. These drawing tools are:

- Select one or more graphic objects.
- Move the selected objects to another location on the screen.
- Scale the selected objects.
- Stretch the selected objects.
- Rotate the selected objects.
- Reshape the selected objects.
- Magnify the selected objects.
- Add text to the drawing.
- Draw a line.
- Draw a multiline.
- Draw an open spline.
- Draw an ellipse.
- Draw a rectangle.
- Draw a polygon.
- Draw a closed spline.

The panel created by **Tools** is made up of panel items known as **IdrawTools**. Each of the above functions is represented by a class inherited from **IdrawTool**. The **IdrawTool** hierarchy is shown in Figure 4.6.

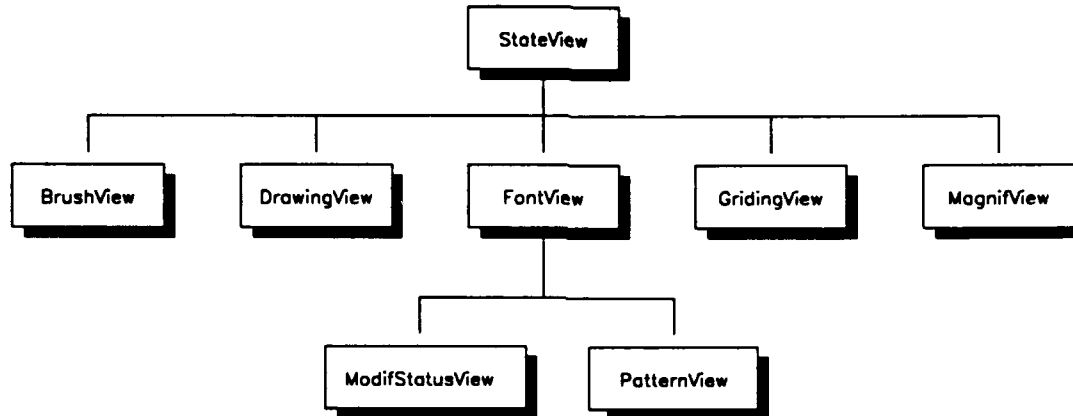


Figure 4.5: StateView Hierarchy

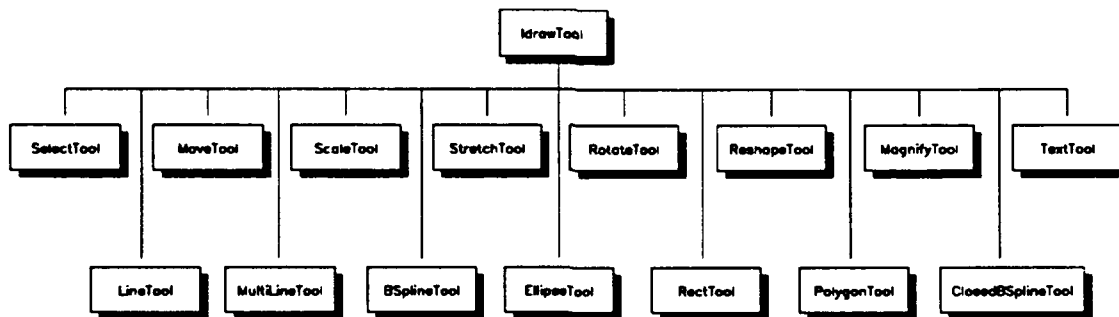


Figure 4.6: IdrawTool Hierarchy

2. Changes

The following is a list of major changes to be made to Idraw to construct the graphic editor:

- Present Prototype Names To The User.
- Remove Unused Commands and Tools.
- Add Internal Representation of DFD.
- Modify Existing Commands and Tools.
- Add New Tools.
- Add Ability to Rebuild DFD Data Structure.
- Add Message Block.

Most of these changes are alterations to existing code. For these, the formal design is not given. For areas where a new design was needed, the formal design is presented.

a. Present Prototype Names To User

Informal Solution

Display prototype names, instead of file names, to the user. Accept a prototype name as input to the graphic editor. To open and save drawings inside the editor, present a list of prototype names and wait for a selection.

Implementation Strategy

The Idraw driver function, **Main**, has one argument, **initial_filename**. Modify **Main** to accept two arguments, **initial_prototype_name** and **prototype_directory**. These are the name of the prototype which contains the DFD to be edited, and the directory where the prototype's DFD file is located. Change Idraw's operation, **ParseArgs**, to incorporate the two arguments into the graphic editor.

Modify the Open and Save operations of the **Editor** class to use instances of the **Selector** class to present the names of prototypes available to the user for editing or saving.

The **Selector** class was presented in the design of the user interface. **Idraw** uses instances of the **FileChooser** class to present file names to the user.

b. Remove Unused Commands and Tools

Informal Solution

The following commands are not needed to edit an enhanced DFD and will be removed:

- **FlipHorizontal**
- **FlipVertical**
- **90Clockwise**
- **90CounterCW**
- **PreciseMove**
- **PreciseScale**
- **PreciseRotate**
- **Group**
- **Ungroup**
- **BringToFront**
- **SendToBack**
- **NumberOfGraphics**
- **New**
- **All patterns except solid, opaque, and transparent**
- **All Brush types.**

The following tools are not needed or replaced by new tools with alter functionality:

- **ScaleTool**
- **StretchTool**
- **RotateTool**
- **MagnifyTool**
- **TextTool**
- **LineTool**
- **MultiLineTool**
- **RectTool**

- **PolygonTool**
- **ClosedBSplineTool.**

Implementation Strategy

To remove the commands and tools not needed to draw a DFD, the pull down menus and the panel items that represent those commands and tools must be removed. The command objects that are removed are a subset of the objects that are instances of the subclasses of the **IdrawCommand** class. The tool objects that are removed are a subset of the objects that are instances of the subclasses of the **IdrawTool** class. The modified inheritance hierarchy of **IdrawTool** is shown in Figure 4.7.

The operations that process the command or tool are a part of the **Editor** class and must also be removed. The operations that are called by a tool start with the word *Handle*. For example, the **HandleSelect** operation is called by the Select tool. The operations that are called by a command are the same name as the command. For example, the **Save** operation is called by the **Save** command.

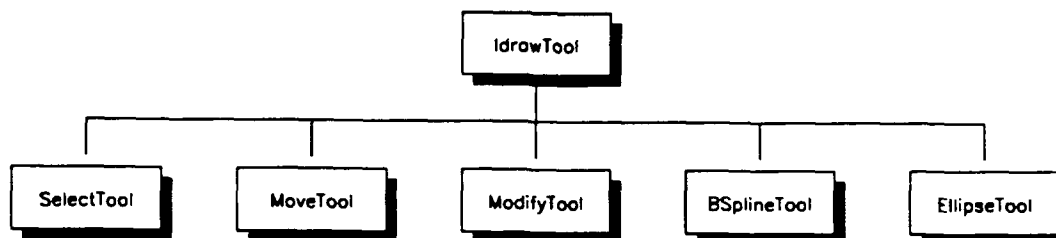


Figure 4.7: Modified IdrawTool Hierarchy

c. Add Internal Representation of DFD

Informal Solution

Create an internal representation of the drawing which will relate data flows, self loops and labels to operators. The PSDL text representing each operator must also be maintained with each operator.

Formal Solution

Class OperatorList

Behavior: Maintains the list of drawn operators and their associated labels, maximum execution times, self loops, and data flows.

Set of Attributes:

operator : OperatorSelection: The ellipse drawn on the screen and its associated graphic objects.

Operations:

OperatorList: Initializes the list.
Remove: Removes an operator from the list.
Add: Add an operator to the list.
Search: Search the list to find the given operator.

Class OperatorSelection

Behavior: Stores all graphic objects drawn in the editor that are related to an operator.

Set of Attributes:

ellipse : EllipseSelection: Ellipse drawn on screen to represent operator.
label : TextSelection: Operator's identifier.
met : TextSelection: Operator's maximum execution time.
input_list : DFDSplineList: List of input data flows associated with operator.
output_list : DFDSplineList: List of output data flows associated with operator.
selfloop_list : DFDSplineList: List of self loops associated with operator.
PSDL_buffer : TextBuffer: Stores current PSDL that represents operator.

The classes EllipseSelection, TextBuffer, and TextSelection are provided with Idraw.

Operations:

OperatorSelection:	Initialize all components and add initial PSDL text to PSDL_buffer .
SetEllipse:	Set value of ellipse .
SetLabel:	Set value of label and add operator's identifier to PSDL_buffer .
SetMET:	Set value of met and add maximum execution time to PSDL_buffer .
AddInput:	Add data flow to input_list and add input place holder to PSDL_buffer .
AddOutput:	Add data flow to output_list and add output place holder to PSDL_buffer .
AddInputLabel:	Add data flow's label to input_list and replace inputplaceholder with label in PSDL_buffer .
AddOutputLabel:	Add data flow's label to output_list and replace output placeholder with label in PSDL_buffer .
AddSelfLoop:	Add self loop to selfloop_list and add state place holder to PSDL_buffer .
AddSelfLoopLabel:	Add self loop's label to selfloop_list and replace state placeholder with label in PSDL_buffer .
RemoveLabel:	Remove label and remove operator's identifier in PSDL_buffer .
RemoveMET:	Remove met and remove maximum execution time in PSDL_buffer .
RemoveInput:	Remove data flow and its label from input_list and remove input data flow and its label from PSDL_buffer .
RemoveOutput:	Remove data flow and its label from output_list and remove output data flow and its label from PSDL_buffer .
RemoveSelfLoop:	Remove data flow and its label from selfloop_list and remove state and its label from PSDL_buffer .
RemoveInputLabel:	Remove data flow's label from input_list and replace input label with input placeholder in PSDL_buffer .
RemoveOutputLabel:	Remove data flow's label from output_list and replace output label with output placeholder in PSDL_buffer .

RemoveSelfLoopLabel:	Remove data flow's label from selfloop_list and replace state label with output placeholder in PSDL_buffer .
-----------------------------	--

Class DFDSplineList

Behavior: Maintains list of splines and their labels.

Set of Attributes:

spline : DFDSplineSelection:	The directed spline and its label and latency drawn on the screen.
-------------------------------------	--

Operations:

DFDSplineList:	Initializes list.
Add:	Add a spline to the list.
Search:	Search the list to find the given spline.

Class DFDSplineSelection

Behavior: Stores spline and its label.

Set of Attributes:

spline : SplineSelection:	Directed spline drawn on the screen to represent the flow of data.
label : TextSelection:	Label associated with spline .
latency : TextSelection:	Maximum firing time associated with data flow.

The classes **SplineSelection** and **TextSelection** are provided with **Idraw**.

Operations:

SetSpline:	Set spline .
SetLabel:	Set label .
SetLatency:	Set latency .

Implementation Strategy

The **OperatorList** class should be a linked list of **OperatorSelections**. An **OperatorSelection** should contain a label, a maximum execution time, zero or more self loops (states) and their labels, and zero or more input data flows, their latencies, and their labels. The self loops and their labels, the inputs and their labels, and the outputs and their

labels should be stored in instances of their **DFDSplineList**, a linked list of **DFDSplineSections**.

Add an instance of the **OperatorList** class to the **Drawing** class as one of its attributes. **Drawing** should provide the interface between the drawing operations done by the user and the maintenance of the list. Add operations to **Drawing** to append drawn objects to the list.

Also add three instances of the **TextBuffer** class to the attributes of **Drawing** to maintain the PSDL specification of the drawing, the data streams added to the drawing, and the control constraints of the drawing. These attributes may be called **streams_buffer**, **constraints_buffer**, and **specification_buffer**. Add operations to **Drawing** to retrieve text from these buffers and add text to these buffers.

d. Modify Existing Commands and Tools

Informal Solution

Because of the new DFD representation, the existing tools and commands must be changed to modify this representation. The changes needed are described below.

Delete — Change the Delete command to also delete the selected objects from the DFD internal representation when deleting them from the drawing. If this deletion creates inconsistency in the stored PSDL text, the PSDL must be updated to reflect the change.

Modify — Modify the Move tool to move an operator's related objects when the operator is moved. If an operator is moved, its label, maximum execution time, self loops, and input and output data flows must also move.

Reshape — Change the functionality of the Reshape tool to modify data flows and self loops and edit text on the screen. Change the name of the tool to be Modify Tool since a data flow's shape is modified by moving its endpoints and text is modified by editing it.

Ellipse — Modify the Ellipse tool to draw an ellipse with a set radius. This will insure that all operators will be the same size in the DFD.

Spline — Change the Spline tool to draw a data flow or self loop to the edge of the operator it is attached to. This should not be assumed to be done by the user. Therefore, if an endpoint of a data flow or self loop is set inside an operator, the endpoint meet automatically be changed to be the intersection of it and the operator.

Implementation Strategy

DeleteCommand — Remove the desired object from the **OperatorList** when it is removed from the drawing. This may mean changing the generated PSDL as well. The operation **Delete** in the **Drawing** class must be modified to accomplish the above changes.

MoveTool — When the user chooses to move a single object, only an operator or one of the text instances can be selected. If a subset of the drawing is chosen, then all chosen objects will move. When an operator is moved, its label, its MET and its set of input and output data flows, their labels and their latencies, must also move. This involves changing the **Move** operation of the **Drawing** class to move these objects automatically.

ReshapeTool — This tool was used to change the shape of a selected object. It will now be used to edit the text on the screen, and change the shape of a data flow or self loop. Its name will be changed to **ModifyTool**. The user must not be allowed to choose an operator with this tool. The **Reshape** operation of the **Drawing** class must change the objects in the **OperatorList** when these modifications are made. Its name must be changed to **Modify**.

EllipseTool — Change the **HandleEllipse** operation in the **Editor** class to draw an ellipse with a fixed size radius.

SplineTool — This tool is changed to determine the endpoints of a spline if the spline is drawn into an operator. Its new endpoint is the intersection point of the spline and operator. This is done by the **HandleSpline** operation in the **Editor** class.

e. Add New Tools

Informal Solution

Add PSDL

Idraw's existing tools do not provide all of the functionality needed to create a DFD and its associated PSDL. New tools will be added to make the graphic editor complete. Add new tools to edit the partial PSDL generated from the drawing. Four components of PSDL will be generated in the graphic editor:

- Specification of an operator.
- The graph of the DFD which describes the vertices and the edges that link the vertices together.
- The data streams and timers of the DFD.
- The control constraints and informal description of the DFD.

All of the above items, except the operator specification, are parts of the PSDL implementation component. The PSDL graph does not need to be edited because it can be completely generated from the DFD.

Use a syntax directed editor to edit the above PSDL components to insure syntactically correct PSDL without making the user remember all of the PSDL syntax.

Add Text Tools

Add new tools to distinguish between the different types of text needed for a DFD. Four tools will replace the Text tool to add the following kinds of text to the DFD:

- The label of any of the DFD components.
- The maximum execution time of an operator.
- The latency time of a data flow.
- Comments made to aid in reading the DFD.

Add Operator Decomposition

Add the ability to decompose an operator into a lower level DFD. This implies that another graphic editor must be executed in order to draw the lower level DFD. If the operators are at their lowest level, this tool must also be able to generate the PSDL implementation of the operator, stating its Ada implementation name.

Implementation Strategy

Figure 4.8 shows the new **IdrawTool** hierarchy, shown in Figure 4.7, with the new tools. The **Editor** class will be changed to add interface operations for each of the tools. The **Drawing** class will be modified to update its components because of changes made by the new tools. Each of the tools are described below.

SpecifyTool — When this tool is chosen, the user will be asked to select an operator or click anywhere else in the drawing. The tool interface will be called to invoke a syntax directed editor to edit the PSDL specification of the chosen operator. When editing is completed, the PSDL specification will be updated in design database, and the graphic editor will get the updated PSDL.

StreamsTool, ConstraintsTool — When this tool is chosen, a window containing a PSDL syntax directed editor will open. The text displayed will be the text stored in the **Drawing's** `streams_buffer` or the text stored in the **Drawing's** `constraints_buffer`. Once editing is complete, the file must be read back into the appropriate text buffer.

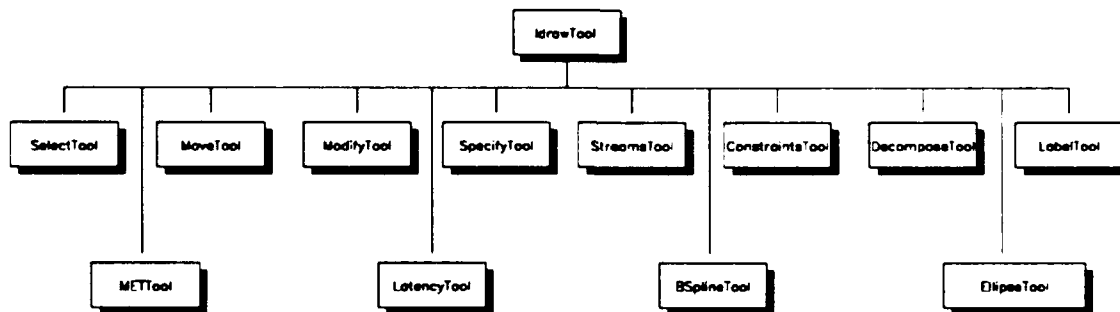


Figure 4.8: New IdrawTool Hierarchy

DecomposeTool — When this tool is chosen, the user should be prompted to select an operator. A dialog box should appear to give the user the choice of decomposing the operator into subcomponents, to search the software base for matching Ada code, or to create the atomic PSDL implementation of the operator. If the user chooses to decompose the operator into suboperators, another CAPS graphic editor should open, and the user can draw a DFD to represent the parent operator. The PSDL that was generated in the calling graphic editor for the decomposed operator should be the same PSDL code that the user sees when he chooses the Specify tool to edit the specification of the drawing.

If the user chooses to create the atomic PSDL implementation of the operator, the following PSDL code should be generated automatically:

```
IMPLEMENTATION ADA <operator_id>
END
```

The <operator_id> should be replaced with the operator's label.

LabelTool — When this tool is chosen, the user should be asked to select an object. The user may then type in the text. The label should be placed in the proper place on the operator and should be added to the **OperatorList**.

METTool — When this tool is chosen, the user should be asked to select an operator. The user can then type in the time and its units. The proper units are milliseconds (ms), seconds (s), minutes (m), or hours (h). The MET should be placed in the proper place on the operator.

LatencyTool — When this tool is chosen, the user should be asked to select a data flow. The user can then type in the maximum allowed delay from the time the data is written into the flow to the time the data can be read from the flow. The proper units are the same ones listed above. The latency should be placed on the data flow.

f. Add Ability to Rebuild DFD Data Structure

Informal Solution

Create a new file when the drawing is saved to write extra information needed to rebuild the DFD data structure. When the drawing is reopened, this new file must be read to rebuild the data structure.

Implementation Strategy

Change the **Save** operation in the **Drawing** class to write out information to represent the **OperatorList** in order to rebuild it when the drawing is edited. Operations are added to the **Drawing** class to accomplish this task. The operation **WriteDFDFiles** will be added to write the following files:

- PSDL implementation file.
- The DFD information file.
- A file for each operator's PSDL specification.

The PSDL implementation file contains the PSDL graph generated from the drawing, the contents of the **streams_buffer**, and the contents of the **constraints_buffer**.

The DFD information file contains each DFD component, its position in the **Drawing**'s list of **Selections (PictSelection)**, and the position in **PictSelection** of the component's operator.

The operator's specification file contains the contents of the operator's **PSDL_buffer**. This file is used by the software base to search for reusable components to match the operator.

Change the **Open** operation in the **Drawing** class to read in these files and add to the proper text buffers when the user chooses to open the drawing.

g. Add Help Information

Informal Solution

Add a message block to display help in how to use each tool. Each tool will display a message in this block when selected. This block may also be used to give help in other areas as needed.

Implementation Strategy

Change the **Idraw** constructor to draw the message block when the editor is drawn on the screen. Change the **Panel** class to store the value of each tool's message and update the message block whenever a new tool is selected. Modify the **State** class to store the current message. Change the **StateView** class to keep track of the user's view of the message.

V. USERS MANUAL

A. INTRODUCTION

The Computer Aided Prototyping System, CAPS, is a software development environment that provides a means to rapidly construct an executable prototype representing a large real-time software system or a software system with hard real-time constraints.

A window-oriented user interface guides the user through the rapid prototyping process. The interface can run on any UNIX workstation that uses the X windowing system.

B. GETTING STARTED

The following environment variables should be set:

- **CAPS:** Directory where all of the CAPS subdirectories can be found.
- **PROTOTYPE:** Directory where prototypes are stored.
- **TEMP:** Directory where temporary files are stored.

The directory \$CAPS/bin must be added to the user's path.

Execute CAPS by typing "caps" on the command line in any X window. By default, its main menu will appear in the upper right corner of the screen. Figure 5.1 shows the CAPS main menu. If the initial menu position is not desirable, it may be repositioned by placing the mouse on the title bar of the menu window and holding down the middle mouse button. Then, move the menu to the desired position and release the mouse button.

The main menu is made up of push buttons. The first five buttons (from left to right) control the functions of the rapid prototyping process: Edit, Search, Translate, Compile, and Execute. The Quit button is used to exit from CAPS.

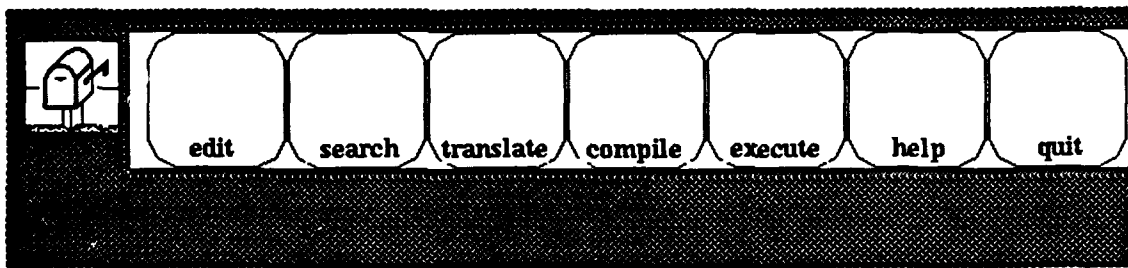


Figure 5.1: CAPS main menu

C. CAPS CONCEPTS

1. General

CAPS is made up of seven tools: graphic editor, software base, design database, translator, static scheduler, dynamic scheduler, and Ada compiler. These tools are used together to rapidly construct the prototype.

The prototyping life cycle begins by the designer drawing or editing a data flow diagram using the graphic editor producing PSDL code. Then, the designer uses the syntax directed editor to create specifications for the operators and types. The designer searches the software base for reusable components to match the PSDL specification of each of the operators. The PSDL is translated into Ada using the translator, static scheduler, and dynamic scheduler. The Ada code is compiled to make an executable prototype. The last step is to execute the prototype. If an error is found or it is found that the prototype is unacceptable at any of these stages, the designer can redo any or all of the steps.

2. Graphic Editor Concepts

The Graphic Editor is used to draw an enhanced DFD and produce a partial PSDL representation of the prototype. Figure 5.2 shows the graphic editor.

An enhanced DFD contains operators (bubbles), data flows (arrows), state variables (self loops), and the text associated with these objects. An operator has a label and may have a maximum execution time. A data flow has a label and may have a latency. Latency is the maximum delay of the data flow. A state variable has an associated label.

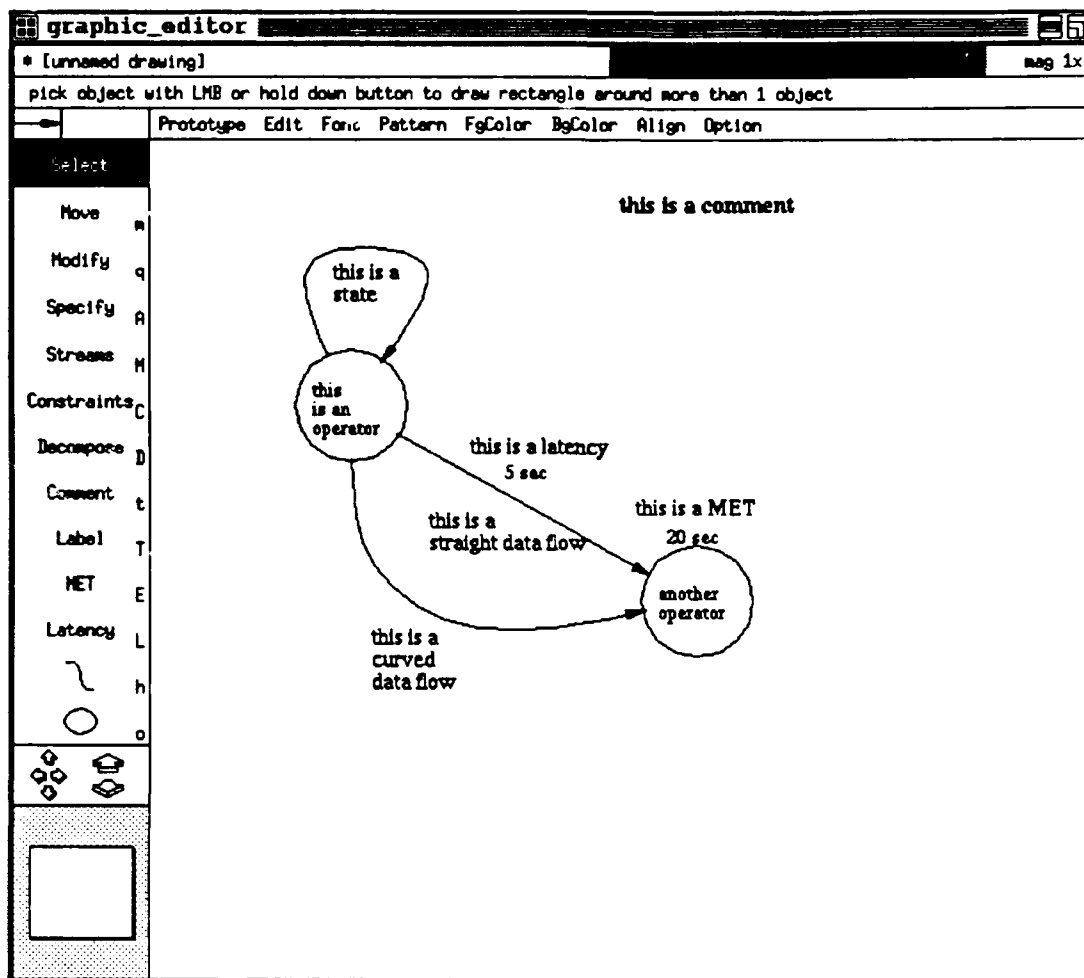


Figure 5.2: Graphic Editor

D. USING CAPS

The user interface is a window and mouse oriented interface. It is assumed that the user is familiar with the X Window environment. Consult the X Windows reference manual for more information. The user can select any of the CAPS functions by placing the mouse pointer on the corresponding menu button and pressing the left mouse button.

After a function has been selected, a pop up prototype selector menu will appear which lists the prototype names available for the chosen function. To select one of the prototypes, place the mouse pointer over the prototype name and click the left mouse button. To create a new prototype, type the new name in the frame underneath the frame containing all of the existing prototype names. Click on the Cancel button and no function will be performed. Click on the Select button and the function will be executed with the chosen prototype in a separate window. Figure 5.3 shows the prototype selector.

To exit CAPS, position the mouse over the quit button and click with the left mouse button. This will terminate the program. If any of the function windows are open, the main menu will not disappear until these windows disappear. CAPS can be iconified by positioning the mouse over the black box containing a white cross located in the title bar of the window and pressing the left mouse button. The main menu will turn into an icon. By pressing the middle mouse button inside the icon, it may be moved and placed anywhere on the screen. The icon can be opened again by positioning the mouse over the icon and clicking the left mouse button.

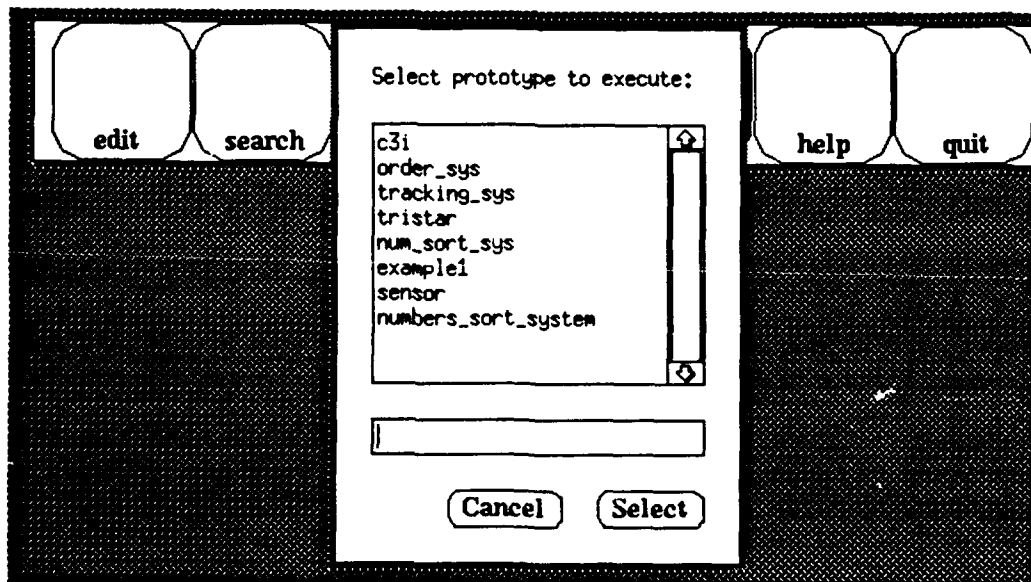


Figure 5.3: Prototype Selector

E. USING THE GRAPHIC EDITOR

1. Graphic Editor User Interface

The graphic editor provides drawing tools and editing commands in order to create or modify an enhanced DFD. The tools are located on the left side of the drawing area, and the commands are displayed as pull down menus at the top of the drawing area.

The tools are run by placing the mouse pointer over the tool button and clicking with the left mouse button. To use the commands, the DFD components to be modified must first be selected using the Select tool. Then, the pull down menu that contains the command is activated by placing the mouse pointer over the menu header and holding down the left mouse button. Drag the mouse pointer to the desired command and release the left mouse button.

2. Tools

a. Select

This tool is used to mark as selected those objects drawn on the screen that will be modified by another tool or command. Select a single graphic object, or select more than one object by holding down the left mouse button and dragging the mouse cursor until each of the desired objects are enclosed in the resulting rectangle.

b. Move

This tool is used to move an operator, text or part of the DFD. A data flow or state cannot be moved by themselves. If an operator moves, its associated objects will move with it. Select a single graphic object, and move the object by holding down the left mouse button and dragging the mouse cursor. If more than one object is to be moved, the object tool must first be selected using the Select tool above.

c. Modify

This tool is used to modify the shape of a data flow or state variable, or edit text. First Select the object to be modified. A data flow or state is then modified by placing the

mouse cursor on the part of the data flow to be modified. Hold down the left mouse button and drag the mouse cursor till the modification is complete. To edit text, place the mouse cursor on the part of the text to be changed and click the left mouse button. Then, edit the text. Text can be added by typing it in, and text can be deleted by pressing the delete key on the keyboard.

d. Specify

This tool is used to edit the PSDL specification of an operator or the entire drawing. Choose an operator by placing the mouse cursor on the operator, or choose the drawing by placing the mouse cursor anywhere else in the drawing area. A window will pop up in the lower right hand portion of the screen. In the current version, the vi editor is used in the window. In the future, this editor will be replaced with a PSDL syntax directed editor. The PSDL specification generated for the operator or drawing will appear in the editor. Text enclosed in "<" and ">" may be replaced with the proper PSDL syntax. WARNING: Changing identifiers or removing inputs, outputs or states while in the editor can result in inconsistencies between the PSDL and the DFD. These should be changed in the drawing, to ensure they will automatically be changed in the PSDL.

e. Streams

This tool is used to edit the PSDL streams and timers associated with the drawing. Streams are those data flows which start in one operator and end in another one. PSDL streams are automatically generated from the DFD. Place the mouse cursor anywhere on the drawing and click with the left mouse button, and a window will pop up containing an editor. The editor contains the PSDL just described. WARNING: Changing identifiers or removing streams while in the editor can result in inconsistencies between the PSDL and the DFD. These should be changed on the drawing, to ensure they will automatically be changed in the PSDL.

f. Constraints

This tool is used to edit PSDL control constraints and informal descriptions. These are not generated from the drawing. Place the mouse cursor anywhere on the drawing and click with the left mouse button, and a window will pop up with an editor.

g. Decompose

This tool is used to decompose an operator into a lower level DFD. First, select an operator. The prototype will automatically be saved. A dialog box will pop up to determine type of decomposition desired. The following choices are available:

- **Graphic Editor:** Another graphic editor will appear and the lower level DFD may be drawn.
- **Ada:** Generate the PSDL declaration for an Ada implementation of the operator. The actual Ada code must be written by the user.
- **Search:** Generate the PSDL declaration for an Ada implementation of the operator and search the software base for a reusable component to match the PSDL specification of the operator.

h. Comment

This tool is used to add comments anywhere on the drawing. It used by placing the mouse cursor where the text is to be located and clicking with the left mouse button, and typing in the text.

i. Label

This tool is used to add a label to any of the DFD components. Choose the desired component, and, then, type in the label. It must be a legal Ada identifier. If it contains blanks or newlines, these will be removed before adding the label to the PSDL representing the operator. When finished, place the mouse cursor outside the DFD components and click with the left mouse button. The label will be centered relative to the chosen component.

j. MET

This tool is used to add a maximum execution time (MET) to an operator. Select the desired operator, and, then type in the MET. It must be an integer followed by one of the following units: *ms*, *sec*, *min*, or *hours*. When finished, click outside the operator and the MET will be placed on top of the operator.

k. Latency

This tool is used to add a latency to a data flow. Latency refers to the maximum possible delay between the time data is written into the stream and the time data is read from the stream. Choose the desired data flow, and, then, type in the latency. It must be an integer followed by one of the following units: *ms*, *sec*, *min*, or *hours*. When finished, click outside the data flow and the latency will be placed on top of the data flow's label. If there is no label, it is placed on the top and centered on the data flow.

l. Data Flow

This tool is used to draw a data flow or state variable. At least one end point of a data flow must be inside of an operator. Both endpoints of a state variable must be inside the same operator. First, choose where the data flow is to begin. If the data flow is to be curved, place the mouse cursor where those curves are to be and click with the left mouse button. Choose where the data flow is to end. If an endpoint is in an operator, the endpoint will automatically be changed to be the intersection of the operator and the spline.

m. Operator

This tool is used to draw an operator. The radius of the operator will be 35 pixels. Choose where the center of the operator is to appear.

3. Commands

Several editing commands are provided via pull down menus. The types of commands available are described below.

a. *Prototype*

This pull down menu present prototype operations. The following operations are available:

- Open a prototype's drawing.
- Save under the existing prototype name.
- Save as a different prototype name.
- Print the prototype's DFD.
- Quit the editor.

b. *Edit*

This pull down menu presents editing operations. The following are available:

- Delete a selected object.
- Select All objects in the drawing.

c. *Font*

This pull down menu presents a list of fonts to be used for the text in the drawing. Only the selected text will use the chosen font.

d. *Pattern*

This pull down menu presents a list of patterns for the selected graphic objects. Only the selected objects will use the chosen pattern.

e. *Color*

This pull down menu presents a list of foreground and background colors for the selected graphic objects.

f. *Align*

This pull down menu presents alignment options such as aligning left sides of selected objects, and aligning centers of selected objects.

g. Options

This pull down menu presents various options to aid in putting the drawing on the page, such as reducing drawing to fit on one page, centering drawing to the page, and providing a grid.

F. USING OTHER CAPS FUNCTIONS

1. Search

This function is used to search the software base for reusable components that match each operator in the data flow diagram drawn in the graphic editor. Work is currently being done to implement the software base. Once the function is chosen, a separate window will appear which contains the message "--- the Search function has not been implemented ---". This window will disappear in 10 seconds. Once the software base is developed, this interaction will change to perform the research.

2. Translate

The PSDL created in the graphic editor is translated into Ada code by the Translate function. Once this function is chosen, a separate window will appear which executes the Translator, Static Scheduler, and the Dynamic Scheduler. Messages will appear in the window indicating which tool is running. Also, any messages produced by the particular tool will also be shown in the window. The window will stay up for 10 seconds after all of these tools have been run to allow the user to finish reading all of the messages provided.

3. Compile

This function is used to compile the reusable components found during the search process and the Ada code produced during the translate process. Once this function is chosen, messages will appear in the window and display messages while the system copies needed files to the current directory and executes the Verdex Ada compiler to compile all

of the reusable components, and the output of the translator, static scheduler, and dynamic scheduler. Messages will appear in the window to inform the user of what is being done. The window will stay up for 10 seconds after the compiler has been run to allow the user to finish reading all of the messages provided.

4. Execute

The prototype is executed by the Execute function. The output of the compilation process is used as input to this function. Once the function is chosen, a separate window will appear which is the actual execution of the prototype. All input and output will be done in this window. The mouse must be in the window to allow any input to be done. To end the execution and make the window disappear, the user must hit the "ctrl" and "c" keys simultaneously.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

The goal of this research is to develop a user interface, tool interface and graphic editor for CAPS. The user interface and tool interface integrates all of the tools of CAPS to provide a user friendly environment. The graphic editor constructs a graphical representation of the prototype.

The user interface guides the user through the rapid prototyping process. It is able to run each of the CAPS tools concurrently and to work with more than one version of a prototype during a session.

The tool interface provides a consistent means of calling the CAPS tools. It provides communication between the CAPS tools and between the user and the CAPS tools while being completely hidden from the user. The creation of the tool interface was a first attempt at developing a manager for the CAPS tools. Some recommendations for its improvement are listed in Section B.

The graphic editor handles all editing of the prototype. It provides many user friendly features, such as moving objects associated with an operator when the operator is moved and updating the PSDL associated with an operator when one of its associated objects is modified or deleted. The graphic editor also automatically generates a PSDL representation of the prototype. The syntax directed editor has not been created for the need of the graphic editor. The current implementation provides "vi" editors instead. The interface to the vi editor has been handled in such a way that replacing vi with another editor can be done easily.

B. RECOMMENDATIONS FOR FURTHER WORK

1. Tool Interface

a. Integrating Interface Functions

Throughout this thesis, the tool interface is described as a separate program. It is currently a set of C functions called from the user interface. These functions create the proper input file names from the given prototype name and fork a process to open an X terminal window to execute the proper tool or tools. If more than one tool is chosen to run, the tool interface builds a shell script to execute them in succession. A separate program needs to be written to implement the tool interface. The design given in Chapter IV should be used to create this program.

b. Integrating the Design Database

A pseudo design database was set up to store the files needed for CAPS in a directory of prototypes. When a design database is implemented, a set of commands should be developed to provide a means of communication between the tool interface and the database. This communication should be oriented toward retrieving and updating files needed by the tools. The following is a list of the files currently needed by the tools. The extensions of the files are shown in parenthesis.

from the graphic editor

- The drawing in post script format (.ps).
- The drawing's PSDL specification (.spec.psdl).
- The drawing's PSDL implementation (.imp.psdl).
- The drawing's graph information (.graph).
- PSDL specification for each of the operators at that level (.op_name.spec.psdl).

When an operator is decomposed in the graphic editor, a new drawing is created. The files stated above are generated for each operator's decomposition.

from the software base

- Reusable components (.sb.a).

If a reusable component cannot be found in the software base to match an operator's specification, the design database should store the Ada component directly written by the designer.

from translator

- Ada translation of prototype (.tl.a).

from static scheduler

- List of composite and atomic operators (.op.info).
- List of atomic operators (.atomic.info).
- List of non-critical operators (.non.crits).
- Ada program which controls execution of operators with timing constraints (.ss.a).

from dynamic scheduler

- Ada program which contains dynamic schedule (.ds.a).

from the compiler

- The executable prototype (.proto).

2. Graphic Editor

The graphic editor was designed to be user friendly and perform as many functions as needed to draw a DFD. There are still some areas that need to be worked on in order to complete the graphic editor.

a. Syntax Directed Editor

A syntax directed editor was created by Laura White [Ref. 5] which supports the complete PSDL grammar. Following the current design, several smaller syntax directed editors should be created and linked into the graphic editor. The tools that need to use these editors are listed below.

- An editor containing the PSDL operator specification should be linked to the Specify tool.
- An editor containing the PSDL streams and timers should be linked to the Streams tool.
- An editor containing the PSDL control constraints and informal description should be linked to the Constraints tool.

It is recommended that the graphic editor call the tool interface to execute these editors. This would provide consistency in the calling of all CAPS tools and would make it easier for these editors to be called from other tools in the future.

b. Consistency Checks

Work needs to be done to perform consistency checks in the graphic editor. If PSDL is changed in a syntax directed editor, it should be checked to make sure that the drawing was not effected. For example, if the identifier of an operator is changed in the PSDL, the label on the operator in the drawing should be changed accordingly. If an input, output, or state is removed in the PSDL, it should be removed from the drawing. Before the drawing is changed in either of these cases, the user should be asked if he really meant for this modification to take place.

When decomposing, the inputs, outputs, and states of the operator must appear in the decomposed picture. These should appear in the picture automatically when the decomposed picture is opened the first time.

c. Modify Tool

Currently, the Modify tool can be used to change the shape of a spline or edit text in the drawing. Another function of this tool should be to move the endpoint of a data flow from:

- one operator to another
- an operator to no operator
- no operator to an operator.

d. Abstract Data Types

The PSDL language provides a PSDL type construct which represents an abstract data type. This language facility is not implemented in the current version of the graphic editor so that the PSDL type component cannot be generated by the graphic editor.

C. CONCLUSIONS

CAPS demonstrates the capability to rapidly construct a prototype of a large real-time system. The changes made to the CAPS user interface as a result of this thesis have made CAPS into a truly usable tool. An example prototype of a Command, Control, Communication and Intelligence System [Ref. 19] has been used to exercise the CAPS tools successfully. Future applications of the CAPS user interface and its interface tools should provide more statistical evidence of productivity improvements and demonstrate the benefits of computer aided design tools in software development.

APPENDIX A

CAPS INTERFACE ESSENTIAL MODEL

ENVIRONMENTAL MODEL

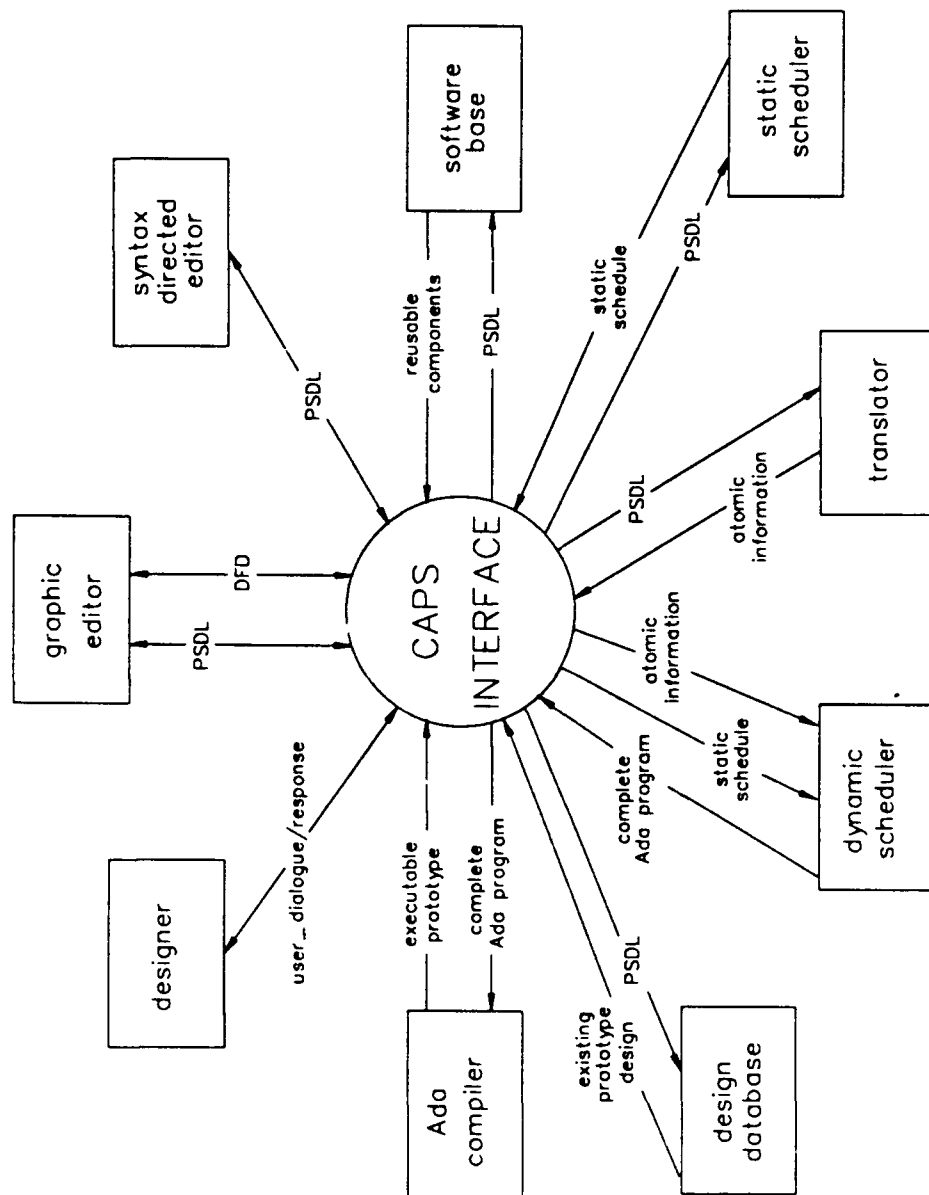
STATEMENT OF PURPOSE

The purpose of the CAPS interface is to establish a software development environment that will provide the designer with a means of constructing a prototype. This interface will allow the designer to have access to a coordinated set of tools that support the development of the prototype.

This environment consists of five levels. The innermost level contains the host operating system. The next level contains X-Windows, a windowing system. Above X-Windows lies InterViews, a user interface toolkit. The next level contains the set of tools that will build the executable prototype. The outer level contains the user interface which provides a user view of all of the tools.

EVENT LIST

- Designer chooses to create a new dataflow diagram.
- Designer chooses to modify a current dataflow diagram.
- Designer chooses to generate PSDL in text mode.
- Designer chooses to generate the dataflow diagram in graphic mode.
- Designer draws the dataflow diagram using the graphical editor.
- Designer creates the PSDL view of the prototype in the syntax directed editor.
- Designer chooses to search for reusable components corresponding to the PSDL view of the prototype.
- Designer chooses to translate the PSDL view of the prototype into Ada code.
- Designer chooses to compile the Ada code for the PSDL view of the prototype.
- Designer chooses to execute the prototype.
- Graphical editor produces part of the PSDL view of the prototype using the graphical representation of the dataflow diagram.
- Syntax-directed editor produces the complete PSDL view of the prototype using the textual representation of the dataflow diagram.
- Software base finds a reusable component based on the PSDL view of the prototype.
- Translator creates an Ada program which contains the atomic description of the PSDL operators.
- Static scheduler creates an Ada program with real time constraints using the PSDL view of the prototype.
- Dynamic scheduler creates the complete Ada program using the outputs from the static scheduler and the translator.
- Ada compiler produces the executable view of the prototype.
- The prototype is executed.
- Design database stores prototype.
- Design database outputs stored prototype.



CAPS INTERFACE CONTEXT DIAGRAM

BEHAVIORAL MODEL

DATA DICTIONARY

ATOMIC INFORMATION = * atomic description of PSDL operators that was constructed using reusable components *

COMPILE = * option to compile code generated by translator which corresponds to prototype *

COMPLETE ADA PROGRAM = **ATOMIC INFORMATION** + **STATIC SCHEDULE**

DFD = * graphic file containing dataflow diagram of prototype *

EDIT = * option to edit prototype *

EDITOR_MENU = * display editor choices to user *

EXECUTABLE PROTOTYPE = * file containing compiled ADA program that is ready for execution *

EXECUTE = * option to execute prototype *

EXISTING PROTOTYPE DESIGN = **DFD** + **PSDL**

GRAPHIC_EDITOR = * option to run graphic editor *

MAIN_MENU = * display CAPS options to user *

MENU_DISPLAYS = **MAIN_MENU** + **EDITOR_MENU** + **PROTOTYPE_NAME_CHOICES_MENU**

PROTOTYPE_NAME = * name of prototype that user will work with *

PROTOTYPE_NAME_CHOICES_MENU = * display prototype names to user from which he can choose *

PSDL = * file containing prototype system design language representation of dataflow diagram *

REUSABLE COMPONENTS = * reusable ADA modules *

SEARCH = * option to search design database for reusable components corresponding to prototype *

SELECTION = [EDIT | SEARCH | TRANSLATE | COMPILE | EXECUTE]

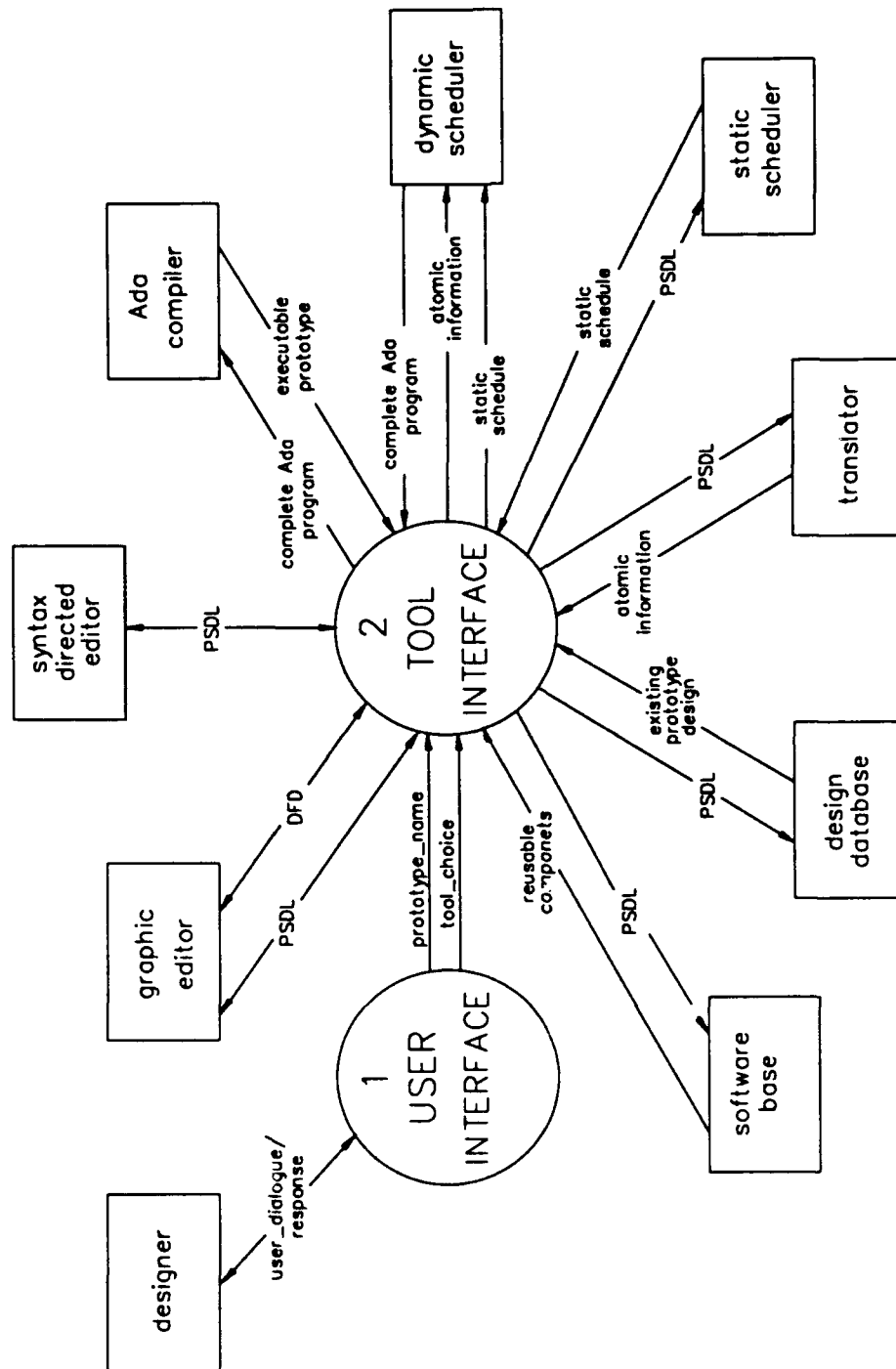
STATIC SCHEDULE = * ADA source code containing procedures and functions describing prototype *

SYNTAX_DIRECTED_EDITOR = * option to run syntax directed editor *

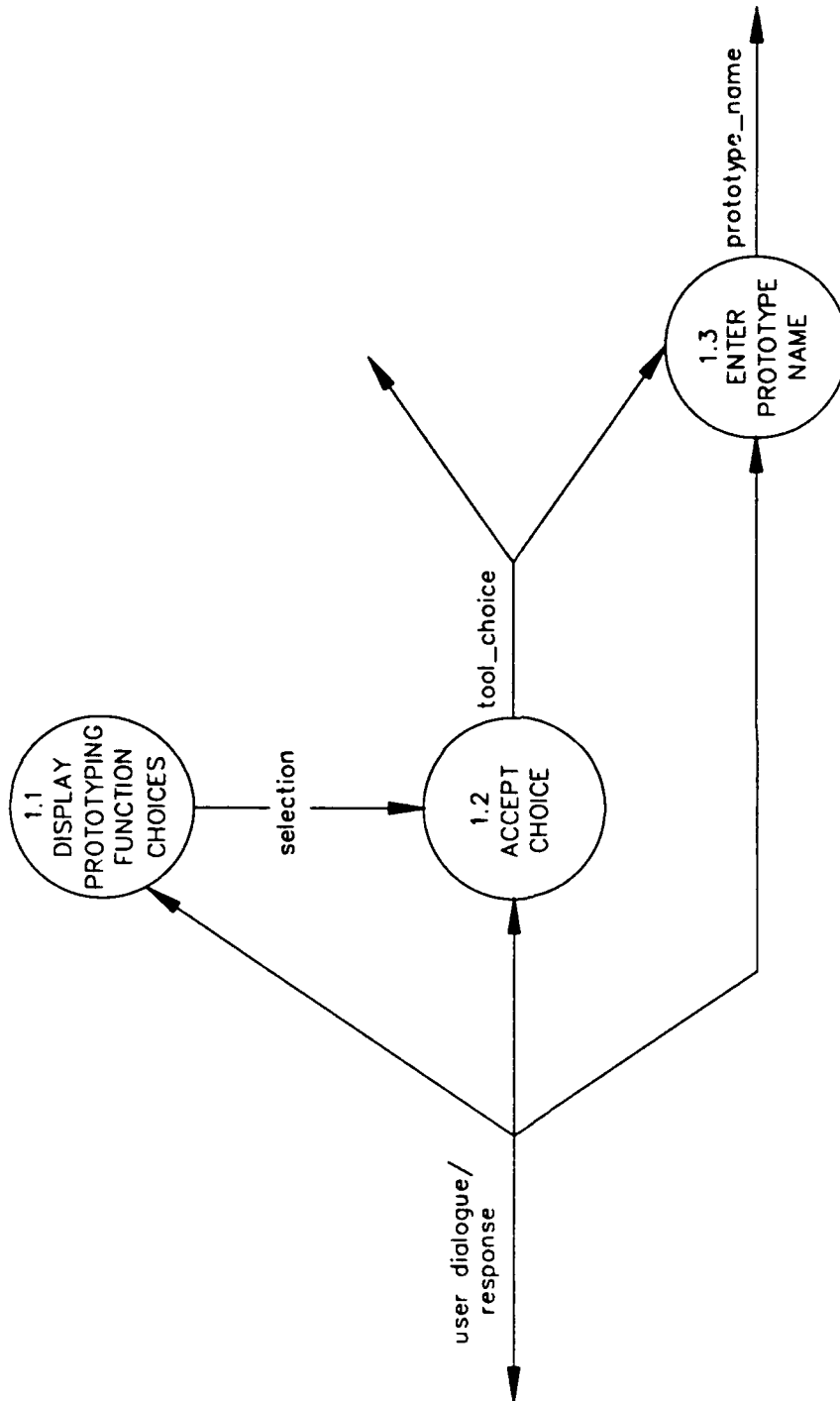
TOOL_CHOICE = [GRAPHIC_EDITOR | SYNTAX_DIRECTED_EDITOR | SEARCH | TRANSLATE | COMPILE | EXECUTE]

TRANSLATE = * option to translate prototype into code *

USER_DIALOGUE/RESPONSE = MENU_DISPLAYS + SELECTION



CAPS INTERFACE DATA FLOW DIAGRAM



USER INTERFACE DATA FLOW DIAGRAM

PROCESS SPECIFICATIONS

PROCESS 1.1: DISPLAY PROTOTYPING FUNCTION CHOICES

BEGIN

 Display **MAIN_MENU**

 set **SELECTION** to menu item chosen

END

PROCESS 1.2: ACCEPT CHOICE

BEGIN

IF SELECTION is **EDIT**

 Display **EDITOR_MENU**

IF GRAPHIC EDITOR chosen

 set **TOOL_CHOICE** to **GRAPHIC_EDITOR**

ELSE

 set **TOOL_CHOICE** to **SYNTAX_DIRECTED_EDITOR**

ENDIF

ELSE

 set **TOOL_CHOICE** to **SELECTION**

END IF

END

PROCESS 1.3: ENTER PROTOTYPE NAME

BEGIN

display **PROTOTYPE_NAME_CHOICES_MENU** depending on

TOOL_CHOICE chosen

set **PROTOTYPE_NAME** to name chosen

END

PROCESS 2: TOOL INTERFACE

BEGIN

PROCESS IN PARALLEL:

TOOL_CHOICE = GRAPHIC_EDITOR:

retrieve **PSDL** file using **PROTOTYPE_NAME**

run **GRAPHIC EDITOR** using **PSDL** file and **DFD** file as input and receiving

PSDL file and **DFD** file as output

TOOL_CHOICE = SYNTAX_DIRECTED_EDITOR:

retrieve **PSDL** file using **PROTOTYPE_NAME**

run **SYNTAX DIRECTED EDITOR** using **PSDL** file as input and receiving

PSDL file as output

TOOL_CHOICE = SEARCH:

retrieve **PSDL** file using **PROTOTYPE_NAME**

run **SOFTWARE BASE** using **PSDL** file as input and receiving **REUSABLE**

COMPONENTS file as output

TOOL_CHOICE = TRANSLATE:

retrieve PSDL file and REUSABLE COMPONENTS file using **PROTO-
TYPE_NAME**

run **TRANSLATOR** using PSDL file and REUSABLE COMPONENTS file as
input and receiving **ATOMIC INFORMATION** file as output

run **STATIC SCHEDULER** using PSDL file and REUSABLE COMPONENTS
file as input and receiving **STATIC SCHECULE** file as output

run **DYNAMIC SCHEDULER** using **ATOMIC INFORMATION** file and
STATIC SCHEDULE file as input and receiving **COMPLETE ADA PRO-
GRAM** file as output

TOOL_CHOICE = COMPILE:

retrieve **COMPLETE ADA PROGRAM** file using **PROTOTYPE_NAME**

run **ADA COMPILER** using **COMPLETE ADA PROGRAM** file as input and
receiving **EXECUTABLE PROTOTYPE** file as output

TOOL_CHOICE = EXECUTE:

retrieve **EXECUTABLE PROTOTYPE** file using **PROTOTYPE_NAME**

run **EXECUTABLE PROTOTYPE**

END IF

END

APPENDIX B

PSDL GRAMMAR

This grammar is provided by Professor Luqi, Naval Postgraduate School in Monterey, California.

Optional items are enclosed in [square brackets]. Items which may appear zero or more times appear in { braces }. Terminal symbols appear in " double quotes ". Groupings appear in (parentheses).

psdl
= { component }

component
= data_type
| operator

data_type
= "type" id type_spec type_impl

type_spec
= "specification" ["generic" type_decl] [type_decl]
{ "operator" id operator_spec }
[functionality] "end"

operator
= "operator" id operator_spec operator_impl

operator_spec
= "specification" { interface } [functionality] "end"

interface
= attribute [reqmts_trace]


```

attribute
    = "generic" type_decl
    | "input" type_decl
    | "output" type_decl
    | "states" type_decl "initially" initial_expression_list
    | "exceptions" id_list
    | "maximum execution time" time

type_decl
    = id_list ":" type_name { "," id_list ":" type_name }

type_name
    = id
    | id "[" type_decl "]"

id_list
    = id { "," id }

reqmts_trace
    = "by requirements" id_list

functionality
    = [keywords] [informal_desc] [formal_desc]

keywords
    = "keywords" id_list

informal_desc
    = "description" "{" text "}"

formal_desc
    = "axioms" "{" text "}"

type_impl
    = "implementation ada" id "end"
    | "implementation" type_name { "operator" id operator_impl } "end"

operator_impl
    = "implementation ada" id "end"
    | "implementation" psdl_impl "end"

psdl_impl
    = data_flow_diagram [streams] [timers] [control_constraints]
    [informal_desc]

```

```

data_flow_diagram
    = "graph" { vertex } { edge }

vertex
    = "vertex" op_id [ ":" time ]
    -- time is the maximum execution time

edge
    = "edge" id [ ":" time ] op_id "->" op_id
    -- time is the latency

op_id
    = id [ "(" [ id_list ] " | " [ id_list ] ")" ]

streams
    = "data stream" type_decl

timers
    = "timer" id_list

control_constraints
    = "control constraints" constraint { constraint }

constraint
    = "operator" op_id
    [ "triggered" [ trigger ] [ "if" expression ] [ reqmts_trace ] ]
    [ "period" time [ reqmts_trace ] ]
    [ "finish within" time [ reqmts_trace ] ]
    [ "minimum calling period" time [ reqmts_trace ] ]
    [ "maximum response time" time [ reqmts_trace ] ]
    { constraint_options }

constraint_options
    = "output" id_list "if" expression [ reqmts_trace ]
    | "exception" id [ "if" expression ] [ reqmts_trace ]
    | timer_op id [ "if" expression ] [ reqmts_trace ]

trigger
    = "by all" id_list
    | "by some" id_list

timer_op
    = "reset timer"
    | "start timer"
    | "stop timer"

```

```

initial_expression_list
    = initial_expression { "," initial_expression }

initial_expression
    = "true"
    | "false"
    | integer_literal
    | real_literal
    | string_literal
    | id
    | type_name "." id ["(" initial_expression_list ")"]
    | "(" initial_expression ")"
    | initial_expression binary_op initial_expression
    | unary_op initial_expression

binary_op
    = "and" | "or" | "xor"
    | "<" | ">" | "=" | ">=" | "<=" | "/"
    | "+" | "-" | "&" | "*" | "/" | "mod" | "rem" | "**"

unary_op
    = "not" | "abs" | "-" | "+"

time
    = integer_literal unit

unit
    = "microsec"
    | "ms"
    | "sec"
    | "min"
    | "hours"

expression_list
    = expression { "," expression }

```

```

expression
    = "true"
    | "false"
    | integer_literal
    | time
    | real_literal
    | string_literal
    | id
    | type_name "." id ["(" initial_expression_list ")"]
    | "(" expression ")"
    | initial_expression binary_op initial_expression
    | unary_op initial_expression

```

```

id
    = letter {alpha_numeric}

```

```

real_literal
    = integer "." integer

```

```

integer_literal
    = digit {digit}

```

```

string_literal
    = "\"" {char} "\""

```

```

char
    = any printable character except "}"

```

```

digit
    = "0 .. 9"

```

```

letter
    = "a .. z"
    | "A .. Z"
    | "_"

```

```

alpha_numeric
    = letter
    | digit

```

```

text
    = {char}

```

APPENDIX C

CAPS INTERFACE PROGRAMMERS MANUAL

A. CAPS INTERFACE FILES

The page numbers listed in the file descriptions indicate where to find these files in this thesis.

1. User Interface

caps_defs.h:	Defines global variables needed by user interface and tool interface. These variables store the environment variables. (p. 92)
caps_main_menu.h:	Class description for caps_main_menu class. (p. 93)
caps_main_menu.c:	Implementation for caps_main_menu class. (p. 94)
main.c:	User interface driver. (p.101)
selector.h:	Class description for Selector class. (p. 102)
selector.c:	Implementation for Selector class. (p. 103)

2. Design Database

design_db.c:	Locates all files that can be used by given function in the given directory and returns them in a array of strings. (p. 99)
---------------------	---

3.Tool Interface

build_scripts.c:	Builds script files needed more than 1 CAPS tool that corresponds to the given function. (p. 87)
tool_interface.c:	Executes CAPS tools based on given function. (p. 106)

B. GUIDELINES FOR FUTURE CHANGES

1. User Interface

The menu is created and displayed in the **Init** function in **caps_main_menu.c**. Each of the push buttons in the menu are created to be .8 inches in size. If this size is changed, the font of the button labels will also need to be changed. This font is defined in **main.c** as an X resource.

The prototype selector dialog box is created and displayed in the **perform_button_function** in **caps_main_menu.c**. It calls the **Selector** constructor to create the box. It calls **find_prototype_names** in **design_database.c** to retrieve the names of the prototypes available for the chosen function. The box is displayed by the **Insert** function in **selector.c**. **Perform_button_function** must be changed when the CAPS design database is implemented.

Perform_button_function also calls the functions in **tool_interface.c** to execute the desired prototyping function. This function must be changed when the tool interface becomes a separate program, instead of a file of functions.

The **Selector** class is based on the **Finder** class in **dialogbox.h** that was supplied with **Idraw**. **Selector** displays a list of strings and should not have to change when the design database is implemented. If it is decided that the **Selector** should display prototype names, and after a name is chosen, it should show all the operators associated with that prototype, the **Insert** function in **selector.c** will have to change.

2. Design Database

The extension of the files that are associated with a function are hard coded in the function **define_extension** in **design_db.c**. The function names are also hard coded in this function.

3. Tool Interface

The functions in **build_scripts.c** build shell scripts for executing CAPS tools. The **make_translate_script** function executes:

- translator
- pre_ss
- decomposer
- static_scheduler
- dynamic_scheduler.

The **make_compile_script** executes **a.make** to compile the **sb.a**, **tl.a**, **ds.a**, **ss.a**, **priority_defs.a**, **vstring_spec.a**, **vstring_body.a**, **timer.a**, **psdl_streams.a**, **glob_dec.a**, and **ds_debugv2.a**. All of these files that are not specific to the given prototype can be found in the translator or static scheduler source directories. All files are copied from its directory to the current directory because the Ada compiler needs them in the current directory in order to compile them.

The functions in **tool_interface.c** use the function system to create an xterm window to execute the tool or tools. The geometry for each of the xterm windows is set in the corresponding function in this file.

C. CODE

```
/*
 * file:          build_scripts.c
 * description:   builds shell scripts to execute translate and
 *               compile functions
 * written by:    Mary Ann Cummings
 */

#include "caps_defs.h"
#include <stdio.h>
#include <string.h>

#define MAX_EXT_LEN 8

void make_filename(char*, char*, char*, const char*);

void make_translate_script(char* prototype_name) {
    FILE* script_file;

/*
 * define character string constants
 */
    char* options = "-o";
    char* translator = "bin/translator";
    char* pre_ss = "bin/pre_ss";
    char* decomposer = "bin/decomposer";
    char* static_scheduler = "bin/static_scheduler";
    char* dynamic_scheduler = "bin/dynamic_scheduler";

    int filename_len = strlen(prototype_dir) + strlen(prototype_name) +
        MAX_EXT_LEN + 1;

    char* script_filename = new char [filename_len];
    char* input_filename = new char [filename_len];
    char* output_filename = new char [filename_len];
    make_filename(script_filename, prototype_dir, prototype_name,
        ".tscript");
    script_file = fopen(script_filename, "w");

/*
 * build translate script file
 */

/*
 * build translator command
 */

    make_filename(input_filename, prototype_dir, prototype_name,
```



```

                                ".psdl");
make_filename(output_filename, prototype_dir, prototype_name,
                                ".tl.a");
fprintf(script_file, "#! /bin/csh\n\n");
fprintf(script_file, "echo --- translating ---\n\n");
fprintf(script_file, "%s%s %s %s %s\n", caps_dir, translator,
        input_filename, options, output_filename);

/*
 * build pre_ss command
 */
make_filename(output_filename, prototype_dir, prototype_name,
        ".op.info");
fprintf(script_file, "\necho --- building static schedule ---\n\n");
fprintf(script_file, "%s%s %s %s %s\n", caps_dir, pre_ss,
        input_filename, options, output_filename);

/*
 * build decomposer command
 */
make_filename(input_filename, prototype_dir, prototype_name,
        ".op.info");
make_filename(output_filename, prototype_dir, prototype_name,
        ".atomic.info");
fprintf(script_file, "\n%s%s %s %s %s\n", caps_dir, decomposer,
        input_filename, options, output_filename);

/*
 * copy file into "atomic.info" because name is hardwired in
 * fp_b.a (this is only a temporary fix)
 */

fprintf(script_file, "cp %s atomic.info\n", output_filename);

/*
 * build static scheduler command
 */
fprintf(script_file, "\n%s%s\n", caps_dir, static_scheduler);

make_filename(output_filename, prototype_dir, prototype_name,
        ".non_crits");
fprintf(script_file, "\ncp non_crits %s\n", output_filename);

make_filename(output_filename, prototype_dir, prototype_name,
        ".ss.a");
fprintf(script_file, "\ncp ss.a %s\n", output_filename);

/*
 * build dynamic scheduler command
 */
fprintf(script_file, "\necho --- building dynamic schedule ---\n\n");
fprintf(script_file, "\n%s%s\n", caps_dir, dynamic_scheduler);

```

```

        make_filename(output_filename, prototype_dir, prototype_name,
                      ".ds.a");
        fprintf(script_file, "\ncp ds.a %s\n", output_filename);
        fprintf(script_file, "\nsleep 10\n");

        fclose(script_file);

/*
 * delete all pointer references
 */
    delete script_file;
    delete output_filename;
    delete input_filename;
    delete script_filename;
}

void make_compile_script(char* prototype_name) {
    FILE* script_file;

/*
 * define character string constants
 */
    char* a_make = "a.make";

    int filename_len = strlen(prototype_dir) + strlen(prototype_name)
                      + MAX_EXT_LEN + 1;

    char* script_filename = new char [filename_len];
    char* output_filename = new char [filename_len];

    make_filename(script_filename, prototype_dir, prototype_name,
                  ".cscript");
    script_file = fopen(script_filename, "w");

    char* static_dir = new char[strlen(caps_dir) + 22];
    strcpy(static_dir, caps_dir);
    strcat(static_dir, "src/static_scheduler/");

    char* trans_dir = new char[strlen(caps_dir) + 16];
    strcpy(trans_dir, caps_dir);
    strcat(trans_dir, "src/translator/");

/*
 * build compile script file
 */
    fprintf(script_file, "#! /bin/csh\n\n");
    fprintf(script_file, "echo --- compiling ---\n\n");

    make_filename(output_filename, prototype_dir, prototype_name,
                  ".sb.a");

```

```

fprintf(script_file, "cp %s sb.a\n", output_filename);

make_filename(output_filename, prototype_dir, prototype_name,
               ".tl.a");
fprintf(script_file, "cp %s tl.a\n", output_filename);

make_filename(output_filename, prototype_dir, prototype_name,
               ".ds.a");
fprintf(script_file, "cp %s ds.a\n", output_filename);

make_filename(output_filename, prototype_dir, prototype_name,
               ".ss.a");
fprintf(script_file, "cp %s ss.a\n", output_filename);

fprintf(script_file, "cp %spriority_defs.a priority_defs.a\n",
        static_dir);
fprintf(script_file, "cp %svstring_spec.a vstring_spec.a\n",
        trans_dir);
fprintf(script_file, "cp %svstring_body.a vstring_body.a\n",
        trans_dir);
fprintf(script_file, "cp %stimer.a timer.a\n", trans_dir);
fprintf(script_file, "cp %spsdl_streams.a psdl_streams.a\n",
        trans_dir);
fprintf(script_file, "cp %sds_debugv2.a ds_debugv2.a\n", trans_dir);
fprintf(script_file, "cp %sglob_dec.a glob_dec.a\n", trans_dir);

char* command = new char [200];

strcpy(command, a_make);
strcat(command, " static_schedule -f ");
strcat(command, "sb.a tl.a ds.a ss.a priority_defs.a
        vstring_spec.a ");
strcat(command, "vstring_body.a timer.a psdl_streams.a
        glob_dec.a ");
strcat(command, "ds_debugv2.a -o ");
strcat(command, prototype_dir);
strcat(command, prototype_name);
strcat(command, ".proto -v");
fprintf(script_file, "%s\n", command);
fprintf(script_file, "sleep 10\n");

fclose(script_file);

/*
 * delete all pointer references
 */
delete script_filename;
delete output_filename;
delete command;
delete static_dir;
delete trans_dir;
delete script_file;

```

```
)  
  
void make_filename(char* result, char* dir, char* prototype_name,  
                  const char* extension) {  
    strcpy(result,dir);  
    strcat(result,prototype_name);  
    strcat(result,extension);  
}
```

```
/*
 * file:      caps_defs.h
 * description: contains values of environment variables as global
 *              variables for CAPS interface.
 * written by: Mary Ann Cummings
 */

#ifndef caps_defs_h
#define caps_defs_h

char* caps_dir;
char* prototype_dir;

#endif
```

```

/*
 * file:          caps_main_menu.h
 * description:   definition of class defining CAPS main menu
 * written by:    Mary Ann Cummings
 */

#ifndef caps_main_menu_h
#define caps_main_menu_h

#include <InterViews/dialog.h>
#include <InterViews/button.h>
#include "selector.h"

class caps_main_menu: public Dialog {
public:
    caps_main_menu(ButtonState*);
    virtual ~caps_main_menu();
    virtual boolean Accept();
private:
    void Init(ButtonState*);
    void AcceptChoice();
    Interactor* AddButtons();
    void perform_button_function(const char*, int);

    PushButton* menu_buttons[7];    // main menu function buttons
    ButtonState* caps_state;        // value of most recent button pushed
};

#endif

```

```

/*
 * file:          caps_main_menu.c
 * description:   implementation of class defining CAPS main menu
 * written by:    Mary Ann Cummings
 */
#include "caps_main_menu.h"
#include "caps_defs.h"
#include <InterViews/shape.h>
#include <InterViews/interactor.h>
#include <InterViews/button.h>
#include <InterViews/event.h>
#include <InterViews/frame.h>
#include <InterViews/glue.h>
#include <InterViews/box.h>
#include <stdio.h>
#include <string.h>

/*
 * main menu button state values
 */
#define NOT_CHOSEN          88
#define EDIT_CHOSEN         1
#define SEARCH_CHOSEN       2
#define TRANSLATE_CHOSEN    3
#define COMPILE_CHOSEN      4
#define EXECUTE_CHOSEN      5
#define HELP_CHOSEN         6

#define MAXPROTOTYPES       100

/*
 * declaration of tool interface routines
 */
void run_editor(char*);
void run_search(char*);
void run_translator(char*);
void run_compiler(char*);
void run_execution(char*);

/*
 * declaration of design database function
 */
void find_prototype_names(char**, char*, const char*);

caps_main_menu::caps_main_menu(ButtonState* quit_state) :
    (quit_state, nil) {
/*
 * constructor for caps_main_menu class
 */
    Init(quit_state);
}

```

```

caps_main_menu::~caps_main_menu() {
/*
 * destructor for caps_main_menu class
 */
    delete caps_state;
}

boolean caps_main_menu::Accept() {
/*
 * loops on each event searching for quit button pushed
 */

    Event e;
    int v;

    state->SetValue(0);
    do {
        Read(e);
        e.target->Handle(e);
        AcceptChoice();
        state->GetValue(v);
    } while (v == 0 && e.target != nil);

    return v == 1 || e.target == nil;
}

void caps_main_menu::Init(ButtonState* quit_state) {
/*
 * want to present dialog box that will ask user for directory that
 * prototypes are stored in
 */

/*
 * initialize all class variables
 */

    caps_state = new ButtonState(NOT_CHOSEN);

    SetClassName("caps_main_menu");

    menu_buttons[0] = new PushButton(" edit ",caps_state,EDIT_CHOSEN);
    menu_buttons[1] = new PushButton(" search ",caps_state,
        SEARCH_CHOSEN);
    menu_buttons[2] = new PushButton("translate",caps_state,
        TRANSLATE_CHOSEN);
    menu_buttons[3] = new PushButton(" compile ",caps_state,
        COMPILE_CHOSEN);
    menu_buttons[4] = new PushButton(" execute ",caps_state,
        EXECUTE_CHOSEN);

```



```

menu_buttons[5] = new PushButton("  help  ",caps_state,HELP_CHOSEN);
menu_buttons[6] = new PushButton("  quit  ",quit_state,true);

/*
 * make each button in the shape of a square
 */
Shape *sh = new Shape();
sh->Square(round(.8*inch));

for (int j = 0; j < 7; ++j)
    menu_buttons[j]->R_shape(*sh);

/*
 * draw main menu on screen
 */
Insert(
    new ShadowFrame(
        new HBox(
            new HGlue(round(0.1*inch), round(0.1*inch)),
            AddButtons()
        )
    )
);
}

void caps_main_menu::AcceptChoice() {
    int val;
    caps_state->GetValue(val);
    switch(val) {
        case EDIT_CHOSEN:
            perform_button_function("edit", 0);
            break;

        case SEARCH_CHOSEN:
            perform_button_function("search", 1);
            break;

        case TRANSLATE_CHOSEN:
            perform_button_function("translate", 2);
            break;

        case COMPILE_CHOSEN:
            perform_button_function("compile", 3);
            break;

        case EXECUTE_CHOSEN:
            perform_button_function("execute", 4);
            break;

        case HELP_CHOSEN:
            menu_buttons[5]->UnChoose();
            caps_state->SetValue(NOT_CHOSEN);
    }
}

```

```

        break;

    default:
        break;
    }
}

Interactor* caps_main_menu::AddButtons() {
/*
 * inserts buttons into main menu
 */
    HBox* box = new HBox();
    box->Align(Center);
    for (int i = 0; i < 7; ++i) {
        box->Insert(menu_buttons[i] );
        box->Insert(new HGlue(round(0.01*inch), round(0.01*inch), 0));
    }
    return box;
}

void caps_main_menu::perform_button_function
    (const char* func_type, int button_number) {
/*
 * define title for prototype selector
 */
    char* edit_select_string = "Select prototype to edit: ";
    char* search_select_string = "Select prototype to search: ";
    char* translate_select_string = "Select prototype to translate: ";
    char* compile_select_string = "Select prototype to compile: ";
    char* execute_select_string = "Select prototype to execute: ";
    char* prototype_name = nil;

/*
 * define dialog box to select prototype
 */
    Selector* sel;
    if (func_type == "edit")
        sel = new Selector(this, edit_select_string, TopCenter);
    else {
        if (func_type == "search")
            sel = new Selector(this, search_select_string, TopCenter);
        else {
            if (func_type == "translate")
                sel = new Selector(this, translate_select_string, TopCenter);
            else {
                if (func_type == "compile")
                    sel = new Selector(this, compile_select_string, TopCenter);
                else {
                    if (func_type == "execute")
                        sel = new Selector(this, execute_select_string, TopCenter);
                }
            }
        }
    }
}

```

```

    }
    }
    char* prototype_array[MAXPROTOTYPES];
    find_prototype_names(prototype_array, prototype_dir, func_type);
    sel->Insert(prototype_array);
    prototype_name = sel->Select();
    delete sel;
/*
 * set main menu button back to white
 */
menu_buttons[button_number]->UnChoose();
caps_state->SetValue(NOT_CHOSEN);

if (prototype_name != nil)
    if (func_type == "edit")
        run_editor(prototype_name);
    else {
        if (func_type == "search")
            run_search(prototype_name);
        else {
            if (func_type == "translate")
                run_translator(prototype_name);
            else {
                if (func_type == "compile")
                    run_compiler(prototype_name);
                else {
                    if (func_type == "execute")
                        run_execution(prototype_name);
                }
            }
        }
    }
delete prototype_name;
}

```

```

/*
 * file:          design_db.c
 * description:   finds the files that are associated with a given
 *               prototype and function, strips off the extension,
 *               and returns the name of the prototypes in an array
 *               of strings.
 * written by:    Mary Ann Cummings
 */

#include <sys/types.h>
#include <sys/dir.h>
#include <string.h>
#include "selecter.h"

int compare_successful(char*, const char*);
char* define_extension(const char*);

void find_prototype_names(char** prototype_array, char* dir,
                          const char* func) {
    DIR* pdir = opendir(dir);
    boolean successful = pdir != NULL;
    struct direct* d;
    char* name;
    int no_of_prototypes = 0;

    if (successful) {
        for (d = readdir(pdir); d != NULL; d = readdir(pdir)) {
            int compare_result = compare_successful(d->d_name, func);
            if (compare_result) {
                name = new char [compare_result + 1];
                strncpy(name, d->d_name, compare_result);
                name[compare_result] = '\0';
                prototype_array[no_of_prototypes] =
                    new char[strlen(name)+1];
                strcpy(prototype_array[no_of_prototypes++], name);
            }
        }
        prototype_array[no_of_prototypes] = nil;
        closedir(pdir);
    }
}

int compare_successful(char* filename, const char* func) {
    char* extension;
    extension = define_extension(func);

    int filename_len = strlen(filename);
    int ext_len = strlen(extension);

    if (filename_len < ext_len)
        return 0;
}

```

```

    else {
        int limit = filename_len - ext_len;
        for (int i = 0; i <= limit; ++i) {
            if (strncmp(filename + i, extension, ext_len) == 0)
                return i;
        }
        return 0;
    }
}

char* define_extension(const char* func) {
    char* extension;
    /*
     * define file extension based on function performed
     */
    if (strcmp(func, "edit") == 0)
        extension = ".spec.psd1";
    else {
        if (strcmp(func, "search") == 0)
            extension = ".imp.psd1";
        else {
            if (strcmp(func, "translate") == 0)
                extension = ".imp.psd1";
            else {
                if (strcmp(func, "compile") == 0)
                    extension = ".sb.a";
                else {
                    if (strcmp(func, "execute") == 0)
                        extension = ".proto";
                }
            }
        }
    }
    return extension;
}

```

```

/*
 * file:          main.c
 * description:   CAPS interface driver.
 * written by:    Mary Ann Cummings
 */
#include <InterViews/button.h>
#include <InterViews/world.h>
#include <stdio.h>
#include <string.h>

#include "caps_main_menu.h"
static PropertyData properties[] = {
    { "caps*PushButton*font",
      "****times-bold-r-normal****140****-iso8859-1"
    },
    { "caps*geometry",
      "-0+0"
    },
    { nil }
};
static OptionDesc options[] = {
    { nil }
};

void GetCapsEnv();

int main (int argc, char* argv[]) {
/*
 * define variables to define CAPS user interface's environment
 */
    World* world = new World("caps", properties, options, argc, argv);
    ButtonState* quit = new ButtonState(false);
    caps_main_menu* cmm = new caps_main_menu(quit);

    cmm->SetName("CAPS main menu");

/*
 * get environment variables
 */

    GetCapsEnv();

/*
 * perform all operations on user interface
 */
    world->InsertApplication(cmm);
    cmm->Accept();           // event loop
    world->Remove(cmm);

    return 0;
}

```

```

/*
 * file:          selector.h
 * description:   class definition of Selector
 * written by:    Mary Ann Cummings
 */

#ifndef selector_h
#define selector_h

#include <InterViews/strchooser.h>
#include <InterViews/frame.h>
#include <InterViews/strbrowser.h>

/*
 * A Selector allows the user to select the prototype he wants to use.
 */

class Selector : public StringChooser {
public:
    Selector(Interactor*, const char*, Alignment);
    char* Select();
    void Insert(char**);
    void SetErrorTitle(const char*);

protected:
    void Init(const char*);
    Interactor* Interior();
    boolean Popup(Event&, boolean = true);

    Interactor* underlying;          // parent interactor that we'll overlay

private:
    Interactor* AddScroller(Interactor*);
    StringBrowser* browser() { return (StringBrowser*) _browser; }

    MarginFrame* title;
    MarginFrame* error_title;
    Alignment align;
};

#endif

```

```

/*
 * file:          selector.c
 * description:   implementation of Selector class
 * written by:    Mary Ann Cummings
 */
#include "selector.h"
#include "istring.h"
#include <InterViews/button.h>
#include <InterViews/event.h>
#include <InterViews/frame.h>
#include <InterViews/message.h>
#include <InterViews/streditor.h>
#include <InterViews/world.h>
#include <InterViews/glue.h>
#include <InterViews/box.h>
#include <InterViews/border.h>
#include <InterViews/adjuster.h>
#include <InterViews/scroller.h>
#include <InterViews/sensor.h>
#include <InterViews/streditor.h>
#include <stdio.h>
#include <string.h>

Selector::Selector (Interactor* u, const char* t, Alignment a) :
    (new ButtonState, 10, 24, "", a) {
    align = a;
    underlying = u;
    Init(t);
    Insert(Interior());
}

char* Selector::Select () {
    char* name = nil;
    Event e;
    if (Popup(e))
        name = Choice();

    return name;
}

void Selector::Insert(char** name_array) {
    for (int i = 0; name_array[i] != nil; ++i)
        browser()->Append(name_array[i]);
}

void Selector::Init(const char* t) {
    if (*t == '\0')
        title = new MarginFrame(new VGlue(0,0));
    else
        title = new MarginFrame(new Message(t));
    error_title = new MarginFrame(new VGlue(0,0));
}

```



```

}

Interactor* Selector::Interior () {
    const int space = round(.5 * cm);
    VBox* errorblock = new VBox(
        new HBox(error_title, new HGlue)
    );

    VBox* titleblock = new VBox(
        new HBox(title, new HGlue)
    );

    return new Frame(
        new MarginFrame(
            new VBox(
                errorblock,
                titleblock,
                new VGlue(space,0),
                new Frame(AddScroller(browser()))),
            new VBox(
                new VGlue(space,0),
                new Frame(new MarginFrame(_sedit,2))
            ),
            new VGlue(space,0),
            new HBox(
                new VGlue(space,0),
                new HGlue,
                new PushButton("Cancel", state, '\007'),
                new HGlue(space,0),
                new PushButton("Select", state, '\r')
            )
        ), space, space/2, 0
    ), 2
    );
}

boolean Selector::Popup (Event&, boolean) {
    World* world = underlying->GetWorld();
    Coord x, y;
    underlying->Align(align, 0, 0, x, y);
    underlying->GetRelative(x, y, world);
    world->InsertTransient(this, underlying, x, y, align);
    boolean accepted = Accept();
    world->Remove(this);
    return accepted;
}

Interactor* Selector::AddScroller(Interactor* i) {
    return new HBox(
        new MarginFrame(i,2),
        new VBorder,
        new VBox(

```

```

        new UpMover(i,1),
        new HBorder,
        new VScroller(i),
        new HBorder,
        new DownMover(i,1)
    )
);
}

void ChangeMsg(const char* name, MarginFrame* frame) {
    Interactor* msg;

    if (*name == '\0')
        msg = new VGlue(0,0);
    else
        msg = new Message(name);
    frame->Insert(msg);
    frame->Change(msg);
}

void Selector::SetErrorTitle(const char* name) {
    ChangeMsg(name, error_title);
}

```

```

/*
 * file:          tool_interface.c
 * description:   executes CAPS tools based on given function.
 * written by:    Mary Ann Cummings
 */
#include "caps_defs.h"
#include <InterViews/defs.h>
#include <string.h>
#include <stdio.h>

char* SetEnvString(char*);

/*
 * get the environment variables needed for caps
 */

void GetCapsEnv() {
    caps_dir = SetEnvString("CAPS");
    prototype_dir = SetEnvString("PROTOTYPE");
}

char* SetEnvString(char* env_const) {
    char* tmp_string;
    int tmp_len;
    char* env_string;
    tmp_string = (char*) getenv(env_const);
    if (tmp_string != nil) {
        tmp_len = strlen(tmp_string);
        env_string = new char[tmp_len + 2];
        strcpy(env_string, tmp_string);
        if (tmp_string[tmp_len - 1] != '/') {
            strcat(env_string, "/");
        }
    }
    else {
        env_string = " ";
    }
    return env_string;
}

void make_translate_script(char*);
void make_compile_script(char*);

char* xterm = "xterm ";
char* in_background = " &";

void run_editor(char* prototype_name) {
    char* my_process = new char [150];

/*
 * make xterm command to execute graphic editor

```

```

*/
strcpy(my_process,caps_dir);
strcat(my_process,"bin/");
strcat(my_process,"graphic_editor");
strcat(my_process," -d ");
strcat(my_process,prototype_dir);
strcat(my_process," -p ");
strcat(my_process,prototype_name);
strcat(my_process,in_background);

system(my_process);

delete my_process;
}

void run_search(char* prototype_name) {
    char* my_process = new char [100];
    char* options = "-T Search -g 60x10+0+150 +sb -e ";

    /*
    * make xterm command to execute search script file
    */
    strcpy(my_process,xterm);
    strcat(my_process,options);
    strcat(my_process,caps_dir);
    strcat(my_process,"bin/search.script");
    strcat(my_process,in_background);

    system(my_process);

    delete my_process;
    delete options;
}

void run_translator(char* prototype_name) {
    make_translate_script(prototype_name);

    /*
    * create command to change file protection so that file can be
    * executed
    */
    char* chmod_command = new char[100];
    strcpy(chmod_command,"chmod 700 ");
    strcat(chmod_command,prototype_dir);
    strcat(chmod_command,prototype_name);
    strcat(chmod_command,".tscript");
    system(chmod_command);

    char* my_process = new char [100];
    char* options = "-T Translator -g 50x10+0+400 +sb -e ";

    /*

```

```

* make xterm command to execute translator script file
*/
strcpy(my_process,xterm);
strcat(my_process,options);
strcat(my_process,prototype_dir);
strcat(my_process,prototype_name);
strcat(my_process,".tscript");
strcat(my_process,in_background);

system(my_process);

delete my_process;
delete chmod_command;
delete options;
}

void run_compiler(char* prototype_name) {
    make_compile_script(prototype_name);

/*
* create command to change file protection so that file can be
* executed
*/
    char* chmod_command = new char[100];
    strcpy(chmod_command,"chmod 700 ");
    strcat(chmod_command,prototype_dir);
    strcat(chmod_command,prototype_name);
    strcat(chmod_command,".cscript");
    system(chmod_command);

    char* my_process = new char [100];
    char* options = "-T Compiler -g +500+550 +sb -e ";

/*
* make xterm command to execute translator script file
*/
    strcpy(my_process,xterm);
    strcat(my_process,options);
    strcat(my_process,prototype_dir);
    strcat(my_process,prototype_name);
    strcat(my_process,".cscript");
    strcat(my_process,in_background);

    system(my_process);

    delete my_process;
    delete chmod_command;
    delete options;
}

void run_execution(char* prototype_name)
{

```

```

char* my_process = new char [100];
char* options = "-T Execute -g +500+400 +sb -e ";

/*
 * make xterm command to execute translator script file
 */
strcpy(my_process,xterm);
strcat(my_process,options);
strcat(my_process,prototype_dir);
strcat(my_process,prototype_name);
strcat(my_process,".proto");
strcat(my_process,in_background);

system(my_process);

delete my_process;
delete options;
}

```

APPENDIX D

GRAPHIC EDITOR PROGRAMMERS MANUAL

A. FILES

The page numbers listed in the file descriptions indicate where to find these files in this thesis. Those files not changed will not have a page number associated with it.

commands.h:	Class description for Commands class (not changed).
commands.c:	Implementation for Commands class and all classes which are commands in the pull down menus. (p. 116)
dfd_defs.h:	Defines all graphic editor-specific variables. (p. 134)
dfdclasses.h:	Sets the value of the class identifiers for all of the DFD objects. (p. 136)
dfdsplinelist.h:	Class description for DFDSplineSelList class. (p. 137)
dfdsplinelist.c:	Implementation for DFDSplineSelList class. (p. 139)
dialogbox.h:	Class description of DialogBox class and its subclasses. (p. 140)
dialogbox.c:	Implementation of DialogBox class and its subclasses. (p. 144)
drawing.h:	Class description of Drawing class. (p. 157)
drawing.c:	Implementation of Drawing class. (p. 162)
drawingview.h:	Class description of DrawingView class (not changed).
drawingview.c:	Implementation of DrawingView class (not changed).
edge.h:	Class description of Edge class. (p. 212)
edge.c:	Implementation of Edge class. (p. 213)
edgelist.h:	Class description of EdgeList class. (p. 215)
edgelist.c:	Implementation of EdgeList class. (p. 217)
editor.h:	Class description of Editor class. (p. 218)
editor.c:	Implementation of Editor class. (p. 222)
errhandler.h:	Class description of ErrHandler class (not changed).
errhandler.c:	Implementation of ErrHandler class (not changed).
highlighter.h:	Class description of Highlighter class (not changed).
highlighter.c:	Implementation of Highlighter class (not changed).

history.h:	Class description of History class (not changed).
history.c:	Implementation of History class (not changed).
idraw.h:	Class description of Idraw class (main class of editor). (p. 257)
idraw.c:	Implementation of Idraw class. (p. 259)
iellipses.h:	Class description of IFillEllipse and IFillCircle (not changed).
iellipse.c:	Implementation of IFillEllipse and IFillCircle classes (not changed).
ipaint.h:	Class description of IBrush, IFont, and IPattern (not changed).
ipaint.c:	Implementation of IBrush, IFont, and IPattern classes (not changed).
isplines.h:	Class description of IFillBSpline and IFillClosedBSpline (not changed).
isplines.c:	Implementation of IFillBSpline and IFillClosedBSpline classes (not changed).
istring.h:	Defines needed string functions. (p. 264)
istring.c:	Implements needed string functions. (p. 265)
keystrokes.h:	Defines keystrokes that map to tools and commands. (p. 267)
list.h:	Class description of BaseList class (not changed).
list.c:	Implementation of BaseList class (not changed).
listboolean.h:	Class description of booleanList class (not changed).
listbrush.h:	Class description of IBrushList class (not changed).
listcenter.h:	Class description of CenterList class (not changed).
listchange.h:	Class description of ChangeList class (not changed).
listchange.c:	Implementation of ChangeList class(not changed).
listcolor.h:	Class description of IColorList class (not changed).
listgroup.h:	Class description of GroupList class (not changed).
listgroup.c:	Implementation of GroupList class (not changed).
listfont.h:	Class description of IFontList class (not changed).
listintrctr.h:	Class description of InteractorList class (not changed).
listipattern.h:	Class description of IPatternList class (not changed).
listselectn.h:	Class description of SelectionList class (not changed).
main.c:	Graphic editor driver. (p. 271)
mapipaint.h:	Class description of MapIPaint and its subclasses (not changed).
mapipaint.c:	Implementation of MapIPaint class and its subclasses (not changed).
mapkey.h:	Class description of MapKey class (not changed).

mapkey.c:	Implementation of MapKey class (not changed).
opsellist.h:	Class description of OperatorSelList. (p. 272)
opsellist.c:	Implementation of OperatorSelList class. (p. 274)
page.h:	Class description of Page class (not changed).
page.c:	Implementation of Page class (not changed).
panel.h:	Class description of Panel and PanelItem (not changed).
panel.c:	Implementation of Panel and PanelItem classes (not changed).
pdmenu.h:	Class description of pull down menu classes (not changed).
pdmenu.c:	Implementation of pull down menu classes (not changed).
rubbands.h:	Class description of IStretchingRect, RubberMultiLine, and RubberPolygon classes (not changed).
rubbands.c:	Implementation of IStretchingRect, RubberMultiLine, and RubberPolygon classes (not changed).
selection.h:	Class description of Selection and NPtSelection (not changed).
selection.c:	Implementation of Selection and NPtSelection classes (not changed).
sldfdspline.h:	Class description of DFDSplineSelection class. (p. 289)
sldfdspline.c:	Implementation of DFDSplineSelection class. (p. 290)
sllellipses.h:	Class description of EllipseSelection and CircleSelection (not changed).
sllellipses.c:	Implementation of EllipseSelection and CircleSelection classes (not changed).
sloperator.h:	Class description for OperatorSelection class. (p. 292)
sloperator.c:	Implementation of OperatorSelection class. (p. 294)
slpict.h:	Class description for PictSelection class (not changed).
slpict.c:	Implementation of PictSelection class (not changed).
slsplines.h:	Class description of BSplineSelection and ClosedBSplineSelection (not changed).
slsplines.c:	Implementation of BSplineSelection and ClosedBSplineSelection classes (not changed).
sltext.h:	Class description of TextSelection (not changed).
sltext.c:	Implementation of TextSelection class (not changed).
state.h:	Class description of State (not changed).
state.c:	Implementation of State class (not changed).
stateviews.h:	Class description of StateView and its subclasses (not changed).
stateviews.c:	Implementation of StateView class and its subclasses (not changed).

textedit.h:	Class description of TextEdit (not changed).
textedit.c:	Implementation of TextEdit class (not changed).
tools.h:	Class description of Tools class (not changed).
tools.c:	Implementation of Tools class and its subclasses. (p. 315)

B. GUIDELINES FOR FUTURE CHANGES

1. PSDL Updates

PSDL is created or modified in the following functions:

in drawing.c

- Drawing
- UpdatePSDLSpec
- WritePSDLGraph
- WriteEdges
- WriteVertices
- AddStream
- AddLabelToStreams
- RemoveStream

in sloperator.c

- OperatorSelection
- AddOperatorIdToPSDL
- AddInputToPSDL
- AddOutputToPSDL
- AddStateToPSDL
- ReplaceInputStringInPSDL
- ReplaceOutputStringInPSDL
- ReplaceStateStringInPSDL
- AddMETToPSDL
- RemoveInputFromPSDL
- RemoveOutputFromPSDL
- RemoveStateFromPSDL

The PSDL keywords are set in the file `dfd_defs.h`. All of the above functions use these keywords to create their portion of the PSDL.

2. Writing Files

All files are written by functions defined in the file `drawing.c`. The function `WriteDFDFiles` creates the following files:

- `<prototype_name>.spec.psdl`: PSDL specification of the drawing.
- `<prototype_name>.imp.psdl`: PSDL implementation of the drawing.
- `<prototype_name>.<op_name>.spec.psdl`: PSDL specification of an operator (done for each operator in the drawing).
- `<prototype_name>.graph`: Drawing information needed to rebuild DFD.

3. Adding New Tools

All tools are associated with a class found in `Tools.c`. For example, the Select tool is created in the `SelectTool` class. Each tool calls a function in `editor.c` to perform the operation associated with the tool. This function begins with the word *Handle*. For example, the `SelectTool` class uses `HandleSelect` to draw the handles on a selected object and add it to the selection list.

4. Adding New Commands

All commands are associated with a class found in `Commands.c`. For example, the Delete command is created in the `DeleteCommand` class. Each command calls a function in `editor.c` to perform the operation associated with the command. The name of the function is the same as the command name. For example, the `DeleteCommand` class uses `Delete` to remove the selected objects from the drawing.

5. Adding New DFD Components

To be able to draw a new object in the DFD, the following will have to be accomplished: Add a new tool to draw the object. Add a new subclass of Selection to represent the object. Add a new Handle function to editor.c to perform the object's operation. Add a new class identifier to dfdclasses.h and add that identifier to the object's class in order to identify the object when needed. Add new functions to opsellist.c and sloperator.c to append the object to the operator list.

6. Adding New X Resources

All X resources are set in main.c. A function must be added to mapipaint.c to retrieve the set resource in order to use it in the editor.

7. Adding Syntax Directed Editors

The vi editor is called by the HandleSpecify, HandleConstraints, and HandleStreams operations in the Editor class. Three variables are defined in dfd_defs.h to map to vi. These variables are: STREAMS_SDE, CONSTRAINTS_SDE, and SPECIFICATION_SDE. To call the syntax directed editors directly from the graphic editor, replace vi with the names of the editors in dfd_defs.h. Otherwise, call the tool interface to call the proper syntax directed editor. If this is done, fork a process to call the tool interface instead of forking a process to execute vi.

C. CODE

```
// file:          commands.c
// description:   Implementation for Commands class and all classes which
//               are commands in the pull down menus. )

/*
 * Copyright (c) 1987, 1988, 1989 Stanford University
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appear in all copies and that
 * both that copyright notice and this permission notice appear in s
 * supporting documentation, and that the name of Stanford not be used in
 * advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. Stanford makes no
 * representations about the suitability of this software for any purpose.
 * It is provided "as is"without express or implied warranty.
 *
 * STANFORD DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
 * IN NO EVENT SHALL STANFORD BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
 * OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
 * WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

// $Header: commands.c,v 1.17 89/10/09 14:47:29 linton Exp $
// implements class Commands.

/* Changes made to conform Idraw into CAPS graphic editor
 *
 * Removed the following commands:
 * FlipHorizontal, FlipVertical, 90Clockwise, 90CounterCW, PreciseMove,
 * PreciseScale, PreciseRotate, Group, Ungroup, BringToFront, SendToBack,
 * NumberofGraphics, BrushCommands, New.
 *
 * Changes made by: Mary Ann Cummings
 * Last change made: August 15, 1990
 */

#include "commands.h"
#include "editor.h"
#include "ipaint.h"
#include "istring.h"
#include "keystrokes.h"
#include "mapipaint.h"
#include "mapkey.h"
```

```

#include "sllines.h"
#include "state.h"
#include <InterViews/box.h>
#include <InterViews/painter.h>
#include <InterViews/sensor.h>
#include <InterViews/shape.h>

// An IdrawCommand enters itself into the MapKey so KeyEvents may be
// mapped to IdrawCommands.

class IdrawCommand : public PullDownMenuCommand {
public:
    IdrawCommand(PullDownMenuActivator*, const char*, char, Editor*,
        MapKey* = nil);
protected:
    Editor* editor;    // handles drawing and editing operations
};

// IdrawCommand passes a printable string representing the given
// character for its key string and enters itself into the character's
// slot in the MapKey.

IdrawCommand::IdrawCommand (PullDownMenuActivator* a, const char* n,
                           char c,
Editor* e, MapKey* mk) : (a, n, mk ? mk->ToStr(c) : "") {
    editor = e;
    if (mk != nil) {
        mk->Enter(this, c);
    }
}

// The following is not needed for a DFD editor

/* ***** Start of Commented Out Code *****

// Each class below encapsulates a label, character, and command.

class NewCommand : public IdrawCommand {
public:
    NewCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "New", NEWCHAR, e, mk) {}
    void Execute (Event&) {
        editor->New();
    }
};

***** End of Commented Out Code ***** */

class RevertCommand : public IdrawCommand {
public:
    RevertCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Revert", REVERTCHAR, e, mk) {}
};

```

```

        void Execute (Event&) {
            editor->Revert();
        }
};

class OpenCommand : public IdrawCommand {
public:
    OpenCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Open...", OPENCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Open();
    }
};

class SaveCommand : public IdrawCommand {
public:
    SaveCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Save", SAVECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Save();
    }
};

class SaveAsCommand : public IdrawCommand {
public:
    SaveAsCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Save As...", SAVEASCHAR, e, mk) {}
    void Execute (Event&) {
        editor->SaveAs();
    }
};

class PrintCommand : public IdrawCommand {
public:
    PrintCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Print...", PRINTCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Print();
    }
};

class QuitCommand : public IdrawCommand {
public:
    QuitCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Quit", QUITCHAR, e, mk) {}
    void Execute (Event& e) {
        editor->Quit(e);
    }
};

class UndoCommand : public IdrawCommand {
public:

```

```

        UndoCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Undo", UNDOCHAR, e, mk) {}
        void Execute (Event&) {
            editor->Undo();
        }
};

class RedoCommand : public IdrawCommand {
public:
    RedoCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "Redo", REDOCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Redo();
    }
};

class CutCommand : public IdrawCommand {
public:
    CutCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "Cut", CUTCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Cut();
    }
};

class CopyCommand : public IdrawCommand {
public:
    CopyCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "Copy", COPYCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Copy();
    }
};

class PasteCommand : public IdrawCommand {
public:
    PasteCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "Paste", PASTECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Paste();
    }
};

class DuplicateCommand : public IdrawCommand {
public:
    DuplicateCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "Duplicate", DUPLICATECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Duplicate();
    }
};

```



```

class DeleteCommand : public IdrawCommand {
public:
    DeleteCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Delete", DELETECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Delete();
    }
};

class SelectAllCommand : public IdrawCommand {
public:
    SelectAllCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Select All", SELECTALLCHAR, e, mk) {}
    void Execute (Event&) {
        editor->SelectAll();
    }
};

// The following code was commented out because these commands were
// not needed in a data flow diagram specific drawing editor

/* ***** Start of Commented Out Code *****

class FlipHorizontalCommand : public IdrawCommand {
public:
    FlipHorizontalCommand (PullDownMenuActivator* a, Editor* e, MapKey*
mk)
        : (a, "Flip Horizontal", FLIPHORIZONTALCHAR, e, mk) {}
    void Execute (Event&) {
        editor->FlipHorizontal();
    }
};

class FlipVerticalCommand : public IdrawCommand {
public:
    FlipVerticalCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Flip Vertical", FLIPVERTICALCHAR, e, mk) {}
    void Execute (Event&) {
        editor->FlipVertical();
    }
};

class _90ClockwiseCommand : public IdrawCommand {
public:
    _90ClockwiseCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "90 Clockwise", _90CLOCKWISECHAR, e, mk) {}
    void Execute (Event&) {
        editor->_90Clockwise();
    }
};

class _90CounterCWCommand : public IdrawCommand {

```

```

public:
    _90CounterCWCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "90 CounterCW", _90COUNTERCWCHAR, e, mk) {}
    void Execute (Event&) {
        editor->_90CounterCW();
    }
};

class PreciseMoveCommand : public IdrawCommand {
public:
    PreciseMoveCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Precise Move...", PRECISEMOVECHAR, e, mk) {}
    void Execute (Event&) {
        editor->PreciseMove();
    }
};

class PreciseScaleCommand : public IdrawCommand {
public:
    PreciseScaleCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Precise Scale...", PRECISESCALECHAR, e, mk) {}
    void Execute (Event&) {
        editor->PreciseScale();
    }
};

class PreciseRotateCommand : public IdrawCommand {
public:
    PreciseRotateCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Precise Rotate...", PRECISEROTATECHAR, e, mk) {}
    void Execute (Event&) {
        editor->PreciseRotate();
    }
};

class GroupCommand : public IdrawCommand {
public:
    GroupCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Group", GROUPCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Group();
    }
};

class UngroupCommand : public IdrawCommand {
public:
    UngroupCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Ungroup", UNGROUPCHAR, e, mk) {}
    void Execute (Event&) {
        editor->Ungroup();
    }
};

```

```

class BringToFrontCommand : public IdrawCommand {
public:
    BringToFrontCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Bring To Front", BRINGTOFRONTCHAR, e, mk) {}
    void Execute (Event&) {
        editor->BringToFront();
    }
};

class SendToBackCommand : public IdrawCommand {
public:
    SendToBackCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Send To Back", SENDTOBACKCHAR, e, mk) {}
    void Execute (Event&) {
        editor->SendToBack();
    }
};

class NumberOfGraphicsCommand : public IdrawCommand {
public:
    NumberOfGraphicsCommand (PullDownMenuActivator* a, Editor* e,
                               MapKey* mk)
        : (a, "Number of Graphics", NUMBEROFGRAPHICSCHAR, e, mk) {}
    void Execute (Event&) {
        editor->NumberOfGraphics();
    }
};

***** End of Commented Out Code ***** */

class FontCommand : public IdrawCommand {
public:
    FontCommand (PullDownMenuActivator* a, Editor* e, IFont* f)
        : (a, f->GetPrintFontAndSize(), '\0', e) {
        font = f;
    }
    void Execute (Event&) {
        editor->SetFont(font);
    }
protected:
    void Reconfig () {
        Font* f = *font;
        if (output->GetFont() != f) {
            Painter* copy = new Painter(output);
            copy->Reference();
            Unref(output);
            output = copy;
            output->SetFont(f);
        }
        IdrawCommand::Reconfig();
    }
    void Resize () { // need constant left pad to line up entries

```

```

    const int xpad = 6;
    name_x = xpad;
    name_y = (ymax - output->GetFont()->Height() + 1) / 2;
    key_x = key_y = 0;
    }
    IFont* font;    // stores font to give Editor
};

static const int PICXMAX = 47; // chosen to minimize scaling for canvas
static const int PICYMAX = 14;

// Brush Command was commented out because only a line with an arrowhead
// on the right end of the line will be used. Broken lines and lines
// without arrowheads are not used in PSDL specific DFD's. It was decided
// to not give the user the choice between which end the arrowhead would
// be placed on because it would complicate the code without giving the
// user much more flexibility

/* ***** Start of Commented Out Code *****

class BrushCommand : public IdrawCommand {
public:
    BrushCommand (PullDownMenuActivator* a, Editor* e, IBrush* b)
        : (a, "None", '\0', e) {
        brush = b;
        brindic = new LineSelection(0, 0, PICXMAX, 0);
        brindic->SetBrush(brush);
        brindic->SetColors(pblack, pwhite);
        brindic->FillBg(true);
        brindic->SetPattern(psolid);
    }
    ~BrushCommand () {
    delete brindic;
    }
    void Execute (Event&) {
    editor->SetBrush(brush);
    }
    void Highlight (boolean on) {
    if (highlighted != on) {
        brindic->SetColors(brindic->GetBgColor(), brindic->GetFgColor());
    }
    IdrawCommand::Highlight(on);
    }
protected:
    void Reconfig () {
    IdrawCommand::Reconfig();
    PColor* fg = brindic->GetFgColor();
    PColor* bg = brindic->GetBgColor();
    if (*fg != output->GetFgColor() || *bg != output->GetBgColor()) {
        fg = new IColor(output->GetFgColor(), "");
        bg = new IColor(output->GetBgColor(), "");
        brindic->SetColors(fg, bg);
    }
}

```

```

    }
    }
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
if (brush->None()) {
    IdrawCommand::Redraw(l, b, r, t);
} else {
    output->ClearRect(canvas, l, b, r, t);
    brindic->Draw(canvas);
}
    }
    void Resize () {
IdrawCommand::Resize();
float xmag = float(xmax - 2*name_x) / PICXMAX;
float hy = float(ymax) / 2;
brindic->SetTransformer(nil);
brindic->Scale(xmag, 1.);
brindic->Translate(float(name_x), hy);
    }
    IBrush* brush;    // stores brush to give Editor
    Graphic* brindic; // displays line to demonstrate brush's effect
};

***** End of Commented Out Code ***** */

class PatternCommand : public IdrawCommand {
public:
    PatternCommand (PullDownMenuActivator* a, Editor* e, IPattern* p,
                    State* s)
        : (a, "None", '\0', e) {
        fgcolor = s->GetFgColor();
        bgcolor = s->GetBgColor();
        pattern = p;
        patindic = nil;
    }
    ~PatternCommand () {
        Unref(patindic);
    }
    void Execute (Event&) {
        editor->SetPattern(pattern);
    }
protected:
    void Reconfig () {
        IdrawCommand::Reconfig();
        if (patindic == nil) {
            patindic = new Painter(output);
            patindic->Reference();
            patindic->SetColors(*fgcolor, *bgcolor);
            patindic->SetPattern(*pattern);
        }
    }
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        if (pattern->None()) {
            IdrawCommand::Redraw(l, b, r, t);
        }
    }

```

```

    } else {
        output->ClearRect(canvas, l, b, r, t);
        patindic->FillRect(canvas, name_x, name_y, xmax-name_x, ymax-name_y);
        output->Rect(canvas, name_x, name_y, xmax-name_x, ymax-name_y);
    }
}

IColor* fgcolor;    // stores initial foreground color
IColor* bgcolor;    // stores initial background color
IPattern* pattern;  // stores pattern to give Editor
Painter* patindic;  // fills rect to demonstrate pat's effect
};

class ColorCommand : public IdrawCommand {
public:
    ColorCommand (PullDownMenuActivator* a, Editor* e, IColor* c)
        : (a, c->GetName(), '\0', e) {
        key = "    ";
        color = c;
        colorindic = nil;
    }
    ~ColorCommand () {
        key = nil;
        Unref(colorindic);
    }
protected:
    void Reconfig () {
        IdrawCommand::Reconfig();
        if (colorindic == nil) {
            colorindic = new Painter(output);
            colorindic->Reference();
            colorindic->SetColors(*color, colorindic->GetBgColor());
        }
    }
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawCommand::Redraw(l, b, r, t);
        colorindic->FillRect(canvas, key_x, key_y, xmax-name_x, ymax-name_y);
        output->Rect(canvas, key_x, key_y, xmax-name_x, ymax-name_y);
    }
    IColor* color;    // stores color to give Editor
    Painter* colorindic; // fills rect to demonstrate color's effect
};

class FgColorCommand : public ColorCommand {
public:
    FgColorCommand (PullDownMenuActivator* a, Editor* e, IColor* c)
        : (a, e, c) {}
    void Execute (Event&) {
        editor->SetFgColor(color);
    }
};

class BgColorCommand : public ColorCommand {

```

```

public:
    BgColorCommand (PullDownMenuActivator* a, Editor* e, IColor* c)
        : (a, e, c) {}
    void Execute (Event&) {
        editor->SetBgColor(color);
    }
};

class AlignLeftSidesCommand : public IdrawCommand {
public:
    AlignLeftSidesCommand (PullDownMenuActivator* a, Editor* e, MapKey*
mk)
        : (a, "Left Sides", ALIGNLEFTSIDESCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignLeftSides();
    }
};

class AlignRightSidesCommand : public IdrawCommand {
public:
    AlignRightSidesCommand (PullDownMenuActivator* a, Editor* e,
MapKey* mk)
        : (a, "Right Sides", ALIGNRIGHTSIDESCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignRightSides();
    }
};

class AlignBottomsCommand : public IdrawCommand {
public:
    AlignBottomsCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Bottoms", ALIGNBOTTOMSCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignBottoms();
    }
};

class AlignTopsCommand : public IdrawCommand {
public:
    AlignTopsCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Tops", ALIGNTOPSCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignTops();
    }
};

class AlignVertCentersCommand : public IdrawCommand {
public:
    AlignVertCentersCommand (PullDownMenuActivator* a, Editor* e,
MapKey* mk)
        : (a, "Vert Centers", ALIGNVERTCENTERSCHAR, e, mk) {}
    void Execute (Event&) {

```

```

        editor->AlignVertCenters();
    }
};

class AlignHorizCentersCommand : public IdrawCommand {
public:
    AlignHorizCentersCommand (PullDownMenuActivator* a, Editor* e,
                               MapKey* mk)
        : (a, "Horiz Centers", ALIGHORIZCENTERSCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignHorizCenters();
    }
};

class AlignCentersCommand : public IdrawCommand {
public:
    AlignCentersCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Centers", ALIGNCENTERSCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignCenters();
    }
};

class AlignLeftToRightCommand : public IdrawCommand {
public:
    AlignLeftToRightCommand (PullDownMenuActivator* a, Editor* e,
                              MapKey* mk)
        : (a, "Left To Right", ALIGNLEFTTORIGHTCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignLeftToRight();
    }
};

class AlignRightToLeftCommand : public IdrawCommand {
public:
    AlignRightToLeftCommand (PullDownMenuActivator* a, Editor* e,
                              MapKey* mk)
        : (a, "Right To Left", ALIGNRIGHTTOLEFTCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignRightToLeft();
    }
};

class AlignBottomToTopCommand : public IdrawCommand {
public:
    AlignBottomToTopCommand (PullDownMenuActivator* a, Editor* e,
                              MapKey* mk)
        : (a, "Bottom To Top", ALIGNBOTTOMTOTOPCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignBottomToTop();
    }
};

```



```

class AlignTopToBottomCommand : public IdrawCommand {
public:
    AlignTopToBottomCommand (PullDownMenuActivator* a, Editor* e,
                             MapKey* mk)
        : (a, "Top To Bottom", ALIGNTOPTOBOTTOMCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignTopToBottom();
    }
};

class AlignToGridCommand : public IdrawCommand {
public:
    AlignToGridCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Align To Grid", ALIGNTOGRIDCHAR, e, mk) {}
    void Execute (Event&) {
        editor->AlignToGrid();
    }
};

class ReduceCommand : public IdrawCommand {
public:
    ReduceCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Reduce", REDUCECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Reduce();
    }
};

class EnlargeCommand : public IdrawCommand {
public:
    EnlargeCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Enlarge", ENLARGECHAR, e, mk) {}
    void Execute (Event&) {
        editor->Enlarge();
    }
};

class NormalSizeCommand : public IdrawCommand {
public:
    NormalSizeCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Normal Size", NORMALSIZECHAR, e, mk) {}
    void Execute (Event&) {
        editor->NormalSize();
    }
};

class ReduceToFitCommand : public IdrawCommand {
public:
    ReduceToFitCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Reduce To Fit", REDUCETOFITCHAR, e, mk) {}
    void Execute (Event&) {

```

```

        editor->ReduceToFit();
    }
};

class CenterPageCommand : public IdrawCommand {
public:
    CenterPageCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Center Page", CENTERPAGECHAR, e, mk) {}
    void Execute (Event&) {
        editor->CenterPage();
    }
};

class RedrawPageCommand : public IdrawCommand {
public:
    RedrawPageCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Redraw Page", REDRAWPAGECHAR, e, mk) {}
    void Execute (Event&) {
        editor->RedrawPage();
    }
};

class GriddingOnOffCommand : public IdrawCommand {
public:
    GriddingOnOffCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Gridding on/off", GRIDDINGONOFFCHAR, e, mk) {}
    void Execute (Event&) {
        editor->GriddingOnOff();
    }
};

class GridVisibleInvisibleCommand : public IdrawCommand {
public:
    GridVisibleInvisibleCommand (PullDownMenuActivator* a, Editor* e,
                                MapKey* mk)
        : (a, "Grid visible/invisible", GRIDVISIBLEINVISIBLECHAR, e, mk) {}
    void Execute (Event&) {
        editor->GridVisibleInvisible();
    }
};

class GridSpacingCommand : public IdrawCommand {
public:
    GridSpacingCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Grid spacing...", GRIDSPACINGCHAR, e, mk) {}
    void Execute (Event&) {
        editor->GridSpacing();
    }
};

class OrientationCommand : public IdrawCommand {
public:

```

```

        OrientationCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
        : (a, "Orientation", ORIENTATIONCHAR, e, mk) {}
        void Execute (Event&) {
            editor->Orientation();
        }
};

class ShowVersionCommand : public IdrawCommand {
public:
    ShowVersionCommand (PullDownMenuActivator* a, Editor* e, MapKey* mk)
    : (a, "", SHOWVERSIONCHAR, e, mk) {
        Listen(noEvents);
    }
    void Execute (Event&) {
        editor->ShowVersion();
    }
protected:
    void Reconfig () {
        shape->width = shape->height = 0;
    }
};

// Commands creates its commands.

Commands::Commands (Editor* e, MapKey* mk, State* s) {
    Init(e, mk, s);
}

// Init creates the activators and commands, inserts the commands into
// menus, gives the menus to the activators, and inserts the activators.

void Commands::Init (Editor* e, MapKey* mk, State* state) {
    PullDownMenuActivator* prot =
        new PullDownMenuActivator(this, "Prototype");
    PullDownMenuActivator* edit =
        new PullDownMenuActivator(this, "Edit");

    // the Structure and Brush pulldown menus were removed because each of
    // their
    // components were not needed in a DFD drawing editor

    /* ***** Start of Commented Out Code *****
    PullDownMenuActivator* strc =
        new PullDownMenuActivator(this, "Structure");
    PullDownMenuActivator* brush =
        new PullDownMenuActivator(this, "brush");
    ***** End of Commented Out Code ***** */

    PullDownMenuActivator* font = new PullDownMenuActivator(this, "Font");
    PullDownMenuActivator* pat =
        new PullDownMenuActivator(this, "Pattern");
    PullDownMenuActivator* fgcolor =

```

```

        new PullDownMenuActivator(this,"FgColor");
PullDownMenuActivator* bgcolor =
        new PullDownMenuActivator(this,"BgColor");
PullDownMenuActivator* align =
        new PullDownMenuActivator(this, "Align");
PullDownMenuActivator* option =
        new PullDownMenuActivator(this, "Option");

Scene* protmenu = new VBox;
// protmenu->Insert(new NewCommand(prot, e, mk));
// protmenu->Insert(new RevertCommand(prot, e, mk));
// protmenu->Insert(new PullDownMenuDivider);
protmenu->Insert(new OpenCommand(prot, e, mk));
protmenu->Insert(new SaveCommand(prot, e, mk));
protmenu->Insert(new SaveAsCommand(prot, e, mk));
protmenu->Insert(new PrintCommand(prot, e, mk));
protmenu->Insert(new PullDownMenuDivider);
protmenu->Insert(new QuitCommand(prot, e, mk));

Scene* editmenu = new VBox;
// editmenu->Insert(new UndoCommand(edit, e, mk));
// editmenu->Insert(new RedoCommand(edit, e, mk));
// editmenu->Insert(new CutCommand(edit, e, mk));
// editmenu->Insert(new CopyCommand(edit, e, mk));
// editmenu->Insert(new PasteCommand(edit, e, mk));
// editmenu->Insert(new DuplicateCommand(edit, e, mk));
editmenu->Insert(new DeleteCommand(edit, e, mk));
editmenu->Insert(new SelectAllCommand(edit, e, mk));

// The following commands were removed from user's view

/* ***** Start of Commented Out Code *****

editmenu->Insert(new PullDownMenuDivider);
editmenu->Insert(new FlipHorizontalCommand(edit, e, mk));
editmenu->Insert(new FlipVerticalCommand(edit, e, mk));
editmenu->Insert(new _90ClockwiseCommand(edit, e, mk));
editmenu->Insert(new _90CounterCWCommand(edit, e, mk));
editmenu->Insert(new PullDownMenuDivider);
editmenu->Insert(new PreciseMoveCommand(edit, e, mk));
editmenu->Insert(new PreciseScaleCommand(edit, e, mk));
editmenu->Insert(new PreciseRotateCommand(edit, e, mk));

Scene* structuremenu = new VBox;
structuremenu->Insert(new GroupCommand(strc, e, mk));
structuremenu->Insert(new UngroupCommand(strc, e, mk));
structuremenu->Insert(new BringToFrontCommand(strc, e, mk));
structuremenu->Insert(new SendToBackCommand(strc, e, mk));
structuremenu->Insert(new PullDownMenuDivider);
structuremenu->Insert(new NumberOfGraphicsCommand(strc, e, mk));
***** End of Commented Out Code ***** */

```

```

    Scene* fontmenu = new VBox;
    MapIFont* mf = state->GetMapIFont();
    for (IFont* f = mf->First(); !mf->AtEnd(); f = mf->Next()) {
        fontmenu->Insert(new FontCommand(font, e, f));
    }

// The following commands were removed

/* ***** Start of Commented Out Code *****

    Scene* brushmenu = new VBox;
    MapIBrush* mb = state->GetMapIBrush();
    for (IBrush* b = mb->First(); !mb->AtEnd(); b = mb->Next()) {
        brushmenu->Insert(new BrushCommand(brush, e, b));
    }
    ***** End of Commented Out Code ***** */

    Scene* patternmenu = new VBox;
    MapIPattern* mp = state->GetMapIPattern();
    for (IPattern* p = mp->First(); !mp->AtEnd(); p = mp->Next()) {
        patternmenu->Insert(new PatternCommand(pat, e, p, state));
    }

    Scene* fgcolormenu = new VBox;
    MapIColor* mfg = state->GetMapIFgColor();
    for (IColor* fg = mfg->First(); !mfg->AtEnd(); fg = mfg->Next()) {
        fgcolormenu->Insert(new FgColorCommand(fgcolor, e, fg));
    }

    Scene* bgcolormenu = new VBox;
    MapIColor* mbg = state->GetMapIBgColor();
    for (IColor* bg = mbg->First(); !mbg->AtEnd(); bg = mbg->Next()) {
        bgcolormenu->Insert(new BgColorCommand(bgcolor, e, bg));
    }

    Scene* alignmenu = new VBox;
    alignmenu->Insert(new AlignLeftSidesCommand(align, e, mk));
    alignmenu->Insert(new AlignRightSidesCommand(align, e, mk));
    alignmenu->Insert(new AlignBottomsCommand(align, e, mk));
    alignmenu->Insert(new AlignTopsCommand(align, e, mk));
    alignmenu->Insert(new AlignVertCentersCommand(align, e, mk));
    alignmenu->Insert(new AlignHorizCentersCommand(align, e, mk));
    alignmenu->Insert(new AlignCentersCommand(align, e, mk));
    alignmenu->Insert(new AlignLeftToRightCommand(align, e, mk));
    alignmenu->Insert(new AlignRightToLeftCommand(align, e, mk));
    alignmenu->Insert(new AlignBottomToTopCommand(align, e, mk));
    alignmenu->Insert(new AlignTopToBottomCommand(align, e, mk));
    alignmenu->Insert(new AlignToGridCommand(align, e, mk));

    Scene* optionmenu = new VBox;
    optionmenu->Insert(new ReduceCommand(option, e, mk));
    optionmenu->Insert(new EnlargeCommand(option, e, mk));

```

```

optionmenu->Insert(new NormalSizeCommand(option, e, mk));
optionmenu->Insert(new ReduceToFitCommand(option, e, mk));
optionmenu->Insert(new CenterPageCommand(option, e, mk));
optionmenu->Insert(new RedrawPageCommand(option, e, mk));
optionmenu->Insert(new PullDownMenuDivider);
optionmenu->Insert(new GriddingOnOffCommand(option, e, mk));
optionmenu->Insert(new GridVisibleInvisibleCommand(option, e, mk));
optionmenu->Insert(new GridSpacingCommand(option, e, mk));
optionmenu->Insert(new OrientationCommand(option, e, mk));
optionmenu->Insert(new ShowVersionCommand(option, e, mk));

prot->SetMenu(protmenu);
edit->SetMenu(editmenu);

/* ***** Start of Commented Out Code *****
strc->SetMenu(structuremenu);
brush->SetMenu(brushmenu);
***** End Of Commented Code ***** */

font->SetMenu(fontmenu);
pat->SetMenu(patternmenu);
fgcolor->SetMenu(fgcolormenu);
bgcolor->SetMenu(bgcolormenu);
align->SetMenu(aligntmenu);
option->SetMenu(optionmenu);

Scene* activators = new HBox;
activators->Insert(prot);
activators->Insert(edit);

/* ***** Start of Commented Out Code *****
activators->Insert(strc);
activators->Insert(brush);
***** End of Commented Out Code ***** */

activators->Insert(font);
activators->Insert(pat);
activators->Insert(fgcolor);
activators->Insert(bgcolor);
activators->Insert(align);
activators->Insert(option);

Insert(activators);
}

// Reconfig makes Commands' shape unstretchable but shrinkable.

void Commands::Reconfig () {
    PullDownMenuBar::Reconfig();
    shape->Rigid(hfil, 0, 0, 0);
}

```

```

// file      dfd_defs.h
// description: Defines all graphic editor-specific variables.

/* Changes made to conform Idraw into CAPS graphic editor:
 * Define values that are needed for the data flow diagram.
 * This header file was created for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last Change made:  August 18, 1990
 */

#ifndef dfd_defs_h
#define dfd_defs_h

#include <InterViews/defs.h>

// pixel radius of operator (ellipse)
#define OperatorRadius 35

// number of characters in PSDL representation of operator
#define TXTBUFLLEN 5000

// max number of prototypes in prototype directory
#define MAXPROTOTYPES 100

// name of scratch file used to edit PSDL for operator
#define PSDL_FILE "psdl.scratch"

// maximum length of message for message block
#define MAXMSGLEN 150

// name of scratch file used to write PSDL streams
#define STREAMS_FILE "streams.scratch"

// name of scratch file used to write PSDL constraints
#define CONSTRAINTS_FILE "constraints.scratch"

// name of file extensions

#define GRAPH_EXT ".ps"
#define GRAPH_EXT_LEN 3
#define IMP_PSDL_EXT ".imp.psdl"
#define IMP_EXT_LEN 9
#define SPEC_PSDL_EXT ".spec.psdl"
#define SPEC_EXT_LEN 10
#define DFD_EXT ".graph"
#define DFD_EXT_LEN 6

// keywords to be inserted when creating PSDL
#define OPER_TKN "OPERATOR "
#define SPEC_TKN "      SPECIFICATION\n"

```

```

#define DESC_TKN "          DESCRIPTION "
#define TEXT_TKN "{ <text> }\n"
#define END_TKN "          END\n"
#define IMP_TKN "          IMPLEMENTATION\n"
#define GR_TKN "          GRAPH\n"
#define VER_TKN "          VERTEX "
#define EDGE_TKN "          EDGE "
#define ID_TKN "<id>"
#define EXT_TKN "EXTERNAL"
#define INPUT_TKN "          INPUT\n"
#define OUTPUT_TKN "          OUTPUT\n"
#define TYPE_DECL_TKN "          <id> : <type_name>"
#define IMP_ADA_TKN "          IMPLEMENTATION ADA "
#define STREAM_TKN "          DATA STREAMS\n"
#define ST_TKN "          STATES <id> : <type_name> initially <expres-
sion>\n"
#define MET_TKN "          MAXIMUM EXECUTION TIME "

// keywords to be used for search through PSDL text buffer.  They are
// different from those above because the text buffer can't ever locate
// newlines
#define INPUT_SCH_TKN "          INPUT"
#define SPEC_SCH_TKN "          SPECIFICATION"
#define OUTPUT_SCH_TKN "          OUTPUT"
#define GEN_SCH_TKN "          GENERIC"
#define STATES_SCH_TKN "          STATES"
#define EXCEPT_SCH_TKN "          EXCEPTIONS"
#define MET_SCH_TKN "          MAXIMUM EXECUTION TIME "
#define MCP_SCH_TKN "          MINIMUM CALLING PERIOD"
#define MRT_SCH_TKN "          MAXIMUM RESPONSE TIME"
#define KEY_SCH_TKN "          KEYWORDS"
#define DESC_SCH_TKN "          DESCRIPTION"
#define AX_SCH_TKN "          AXIOM"
#define END_SCH_TKN "          END"
#define STREAM_SCH_TKN "DATA STREAMS"
#define TIMER_SCH_TKN "TIMER"
#define CON_SCH_TKN "CONTROL CONSTRAINTS"

#define STREAMS_SDE "vi"
#define CONSTRAINTS_SDE "vi"
#define SPECIFICATION_SDE "vi"

#endif

```



```

// file          dfdclasses.h
// description:  Sets the value of the class identifiers for all of the
                DFD objects.

/* Changes made to conform Idraw into CAPS graphic editor:
 * Define class identifiers for each of the DFD components in order to
 * identify what selection is being manipulated.
 * Changed class id TEXT to be LABEL_OP, LABEL_DF, LABEL_SL, or COMMENT
 * in order to tell what type of text we have.
 * This header file was made specifically for graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  September 3, 1990
 */

#ifndef dfdclasses_h
#define dfdclasses_h

static const int OPERATOR          = 2050;
// static const int DATAFLOW_LINE = 2051;
static const int DATAFLOW_SPLINE = 2052;
static const int SELFLOOP          = 2053;
static const int LABEL_OP          = 2054;
static const int LABEL_DF          = 2055;
static const int LABEL_SL          = 2056;
static const int COMMENT           = 2057;
static const int MET_OP            = 2058;
static const int LAT_DF            = 2059;
static const int NONE              = 2099;

#endif

```

```

// file      dfdsplinelist.h
// description: Class description for DFDSplineSelList class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add definition of DFD spline selection class.
 * This file was created specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 22, 1990
 */

#ifndef dfdsplinelist_h
#define dfdsplinelist_h

#include "list.h"

// declare imported classes

class BSplineSelection;
class DFDSplineSelection;

// This class defines a node to be contained in the dfd spline
// selection list

class DFDSplineSelNode : public BaseNode {
public:
    DFDSplineSelNode(DFDSplineSelection* dss) { dfdsplsel = dss; }
    boolean SameValueAs(void* p) { return dfdsplsel == p; }
    DFDSplineSelection* GetSelection() { return dfdsplsel; }

protected:
    DFDSplineSelection* dfdsplsel;    // points to a DFD spline
                                     // selection
};

// This class defines a list of spline selections

class DFDSplineSelList: public BaseList {
public:
    DFDSplineSelNode* First();
    DFDSplineSelNode* Last();
    DFDSplineSelNode* Prev();
    DFDSplineSelNode* Next();
    DFDSplineSelNode* GetCur();
    void SetCur(BSplineSelection*);
    DFDSplineSelNode* Index(int);
};

inline DFDSplineSelNode* DFDSplineSelList::First() {
    return (DFDSplineSelNode*) BaseList::First();
}

```

```

inline DFDSplineSelNode* DFDSplineSelList::Last() {
    return (DFDSplineSelNode*) BaseList::Last();
}

inline DFDSplineSelNode* DFDSplineSelList::Prev() {
    return (DFDSplineSelNode*) BaseList::Prev();
}

inline DFDSplineSelNode* DFDSplineSelList::Next() {
    return (DFDSplineSelNode*) BaseList::Next();
}

inline DFDSplineSelNode* DFDSplineSelList::GetCur() {
    return (DFDSplineSelNode*) BaseList::GetCur();
}

inline DFDSplineSelNode* DFDSplineSelList::Index(int index) {
    return (DFDSplineSelNode*) BaseList::Index(index);
}

#endif

```

```

// file      dfdsplinelist.c
// description: Implementation for DFDSplineSelList class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add implementation of DFD spline selection list class.
 * This file was created specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 22, 1990
 */

#include "dfdsplinelist.h"
#include "sldfdspline.h"
#include "slsplines.h"

// SetCur searches the list of dfd spline selections to make the current
// node be the one that matches the given spline selection

void DFDSplineSelList::SetCur(BSplineSelection* ss) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetSplineSelection() == ss)
            return;
    }
    return;
}

```

```

//file          dialgobox.h
// description:  Class description of DialogBox class and its subclasses.

// $Header: dialogbox.h,v 1.11 89/10/09 14:47:47 linton Exp $
// declares class DialogBox and DialogBox subclasses.

/* Changes made to conform Idraw to CAPS graphic editor;
 * Add new subclass Chooser to display a dialog box of 3 button choices
 * and a cancel button.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 16, 1990
 */

#ifndef dialogbox_h
#define dialogbox_h

#include <InterViews/filechooser.h>

// Declare imported types.

class ButtonState;
class IMessage;
class StringEditor;

// A DialogBox knows how to set its message and warning text and how
// to pop up itself over the underlying Interactor.

class DialogBox : public MonoScene {
public:

    void SetMessage(const char* = nil, const char* = nil);
    void SetWarning(const char* = nil, const char* = nil);
    void SetUnderlying(Interactor*);

protected:

    DialogBox(Interactor*, const char* = nil);

    void PopUp();
    void Disappear();

    IMessage* message;    // displays message text
    IMessage* warning;    // displays warning text
    Interactor* underlying; // we'll insert ourselves into its parent

};

// A Messenger displays a message until it's acknowledged.

class Messenger : public DialogBox {

```

```

public:

    Messenger(Interactor*, const char* = nil);
    ~Messenger();

    void Display();

protected:

    void Init();
    void Reconfig();

    ButtonState* ok;    // stores status of "ok" button
    Interactor* okbutton; // displays "ok" button

};

// A Confirmer displays a message until it's confirmed or cancelled.

class Confirmer : public DialogBox {
public:

    Confirmer(Interactor*, const char* = nil);
    ~Confirmer();

    char Confirm();

protected:

    void Init();
    void Reconfig();

    ButtonState* yes;    // stores status of "yes" button
    ButtonState* no;     // stores status of "no" button
    ButtonState* cancel; // stores status of "cancel" button
    Interactor* yesbutton; // displays "yes" button
    Interactor* nobutton; // displays "no" button
    Interactor* cancelbutton; // displays "cancel" button

};

// A Namer displays a string until it's edited or cancelled.

class Namer : public DialogBox {
public:

    Namer(Interactor*, const char* = nil);
    ~Namer();

    char* Edit(const char*);

protected:

```

```

void Init();
void Reconfig();

ButtonState* accept; // stores status of "accept" button
ButtonState* cancel; // stores status of "cancel" button
Interactor* acceptbutton; // displays "accept" button
Interactor* cancelbutton; // displays "cancel" button
StringEditor* stringeditor; // displays and edits a string

};

// A Finder browses the file system and returns a file name.

class Finder : public FileChooser {
public:

    Finder(Interactor*, const char*);

    const char* Find();

protected:

    Interactor* Interior();
    boolean Popup(Event&, boolean = true);

protected:

    Interactor* underlying; // we'll insert ourselves into its parent

};

// A Chooser displays a set of choices and are displayed until one is
// chosen or is cancelled.

class Chooser : public DialogBox {
public:

    Chooser(Interactor*, const char*, const char*, const char*,
            const char*);

    ~Chooser();

    char Choose();

protected:

    void Init();
    void Reconfig();

    ButtonState* bs_1; // stores status of first button
    ButtonState* bs_2; // stores status of second button
    ButtonState* bs_3; // stores status of third button

```

```
    ButtonState* cancel; // stores status of "cancel" button
    Interactor* button_1; // displays first button
    Interactor* button_2; // displays second button
    Interactor* button_3; // displays third button
    Interactor* cancelbutton; // displays "cancel" button

};
#endif
```



```

// file      dialogbox.c
// description: Implementation of DialogBox class and its subclasses.

/*
 * $Header: dialogbox.c,v 1.17 89/10/09 14:47:45 linton Exp $
 * implements class DialogBox and DialogBox subclasses.
 */

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add implementation of Chooser class to allow user to choose from
 * 3 button choices and a cancel button.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 16, 1990
 */

#include "dialogbox.h"
#include "istring.h"
#include <InterViews/box.h>
#include <InterViews/button.h>
#include <InterViews/canvas.h>
#include <InterViews/event.h>
#include <InterViews/font.h>
#include <InterViews/frame.h>
#include <InterViews/glue.h>
#include <InterViews/message.h>
#include <InterViews/painter.h>
#include <InterViews/sensor.h>
#include <InterViews/shape.h>
#include <InterViews/streditor.h>
#include <InterViews/world.h>

#include <InterViews/Std/os/fs.h>
#include <sys/param.h>

/*
 * An IMessage displays its own text, not somebody else's.
 */

class IMessage : public Message {
public:
    IMessage(const char* = nil, Alignment a = Center);
    ~IMessage();

    void SetText(const char* = nil, const char* = nil);
protected:
    char* buffer;    /* stores own copy of text */
};

/*
 * IMessage creates a buffer to store its own copy of the text.
 */

```

```

*/

IMessage::IMessage (const char* msg, Alignment a) : (nil, a) {
    buffer = strdup(msg ? msg : "");
    text = buffer;
}

/*
 * Free storage allocated for the text buffer.
 */

IMessage::~IMessage () {
    delete buffer;
}

/*
 * SetText stores the new text and changes the IMessage's shape to fit
 * the new text's width.
 */

void IMessage::SetText (const char* beg, const char* end) {
    beg = beg ? beg : "";
    end = end ? end : "";
    delete buffer;
    buffer = new char[strlen(beg) + strlen(end) + 1];
    strcpy(buffer, beg);
    strcat(buffer, end);
    text = buffer;
    if (canvas != nil && canvas->Status() == CanvasMapped) {
        Reconfig();
        Parent()->Change(this);
    }
}

/*
 * DialogBox creates two IMessages to display a message and a warning
 * and stores its underlying Interactor. DialogBox won't delete the
 * IMessages so its derived classes can put them in boxes which will
 * delete them when the boxes are deleted.
 */

DialogBox::DialogBox (Interactor* u, const char* msg) {
    SetCanvasType(CanvasSaveUnder); /* speed up expose redrawing if
                                     possible */

    input = allEvents;
    input->Reference();
    message = new IMessage(msg);
    warning = new IMessage;
    underlying = u;
}

/*

```

```

    * SetMessage sets the message's text.
    */

void DialogBox::SetMessage (const char* beg, const char* end) {
    message->SetText(beg, end);
}

/*
    * SetWarning sets the warning's text.
    */

void DialogBox::SetWarning (const char* beg, const char* end) {
    warning->SetText(beg, end);
}

/*
    * SetUnderlying sets the underlying Interactor over which the
    * DialogBox will pop up itself.
    */

void DialogBox::SetUnderlying (Interactor* u) {
    underlying = u;
}

/*
    * PopUp pops up the DialogBox centered over the underlying
    * Interactor's canvas.
    */

void DialogBox::PopUp () {
    World* world = underlying->GetWorld();
    Coord x, y;
    underlying->Align(Center, 0, 0, x, y);
    underlying->GetRelative(x, y, world);
    world->InsertTransient(this, underlying, x, y, Center);
}

/*
    * Disappear removes the DialogBox. Since the user should see
    * warnings only once, Disappear clears the warning's text so the next
    * PopUp won't display it.
    */

void DialogBox::Disappear () {
    Parent()->Remove(this);
    SetWarning();
    Sync();
}

/*
    * Messenger creates its button state and initializes its view.
    */

```

```

Messenger::Messenger (Interactor* u, const char* msg) : (u, msg) {
    ok = new ButtonState(false);
    okbutton = new PushButton(" OK ", ok, true);
    Init();
}

/*
 * Free storage allocated for the message's button state.
 */

Messenger::~Messenger () {
    Unref(ok);
}

/*
 * Display pops up the Messenger and removes it when the user
 * acknowledges the message.
 */

void Messenger::Display () {
    ok->SetValue(false);

    PopUp();

    int okay = false;
    while (!okay) {
        Event e;
        Read(e);
        if (e.eventType == KeyEvent && e.len > 0) {
            switch (e.keystring[0]) {
                case '\r': /* CR */
                case '\007': /* ^G */
                    ok->SetValue(true);
                    break;
                default:
                    break;
            }
        } else if (e.target == okbutton) {
            e.target->Handle(e);
        }
        ok->GetValue(okay);
    }

    Disappear();
}

/*
 * Init composes Messenger's view with boxes, glue, and frames.
 */

void Messenger::Init () {

```

```

        SetClassName("Messenger");

        VBox* vbox = new VBox;
        vbox->Align(Center);
        vbox->Insert(new VGlue);
        vbox->Insert(warning);
        vbox->Insert(new VGlue);
        vbox->Insert(message);
        vbox->Insert(new VGlue);
        vbox->Insert(okbutton);
        vbox->Insert(new VGlue);

        Insert(new Frame(vbox, 2));
    }

    /*
     * Reconfig pads Messenger's shape to make the view look less crowded.
     */

void Messenger::Reconfig () {
    DialogBox::Reconfig();
    Font* font = output->GetFont();
    shape->width += 2 * font->Width("mmmmmm");
    shape->height += 4 * font->Height();
}

    /*
     * Confirmer creates its button states and initializes its view.
     */

    Confirmer::Confirmer (Interactor* u, const char* prompt) : (u, prompt) {
        yes      = new ButtonState(false);
        no       = new ButtonState(false);
        cancel    = new ButtonState(false);
        yesbutton = new PushButton(" Yes ", yes, true);
        nobutton  = new PushButton(" No  ", no, true);
        cancelbutton = new PushButton("Cancel", cancel, true);
        Init();
    }

    /*
     * Free storage allocated for the button states.
     */

    Confirmer::~Confirmer () {
        Unref(yes);
        Unref(no);
        Unref(cancel);
    }

    /*
     * Confirm pops up the Confirmer, lets the user confirm the message or

```

```

    * not, removes the Confirmer, and returns the confirmation.
    */

char Confirmer::Confirm () {
    yes->SetValue(false);
    no->SetValue(false);
    cancel->SetValue(false);

    PopUp();

    int confirmed = false;
    int denied = false;
    int cancelled = false;
    while (!confirmed && !denied && !cancelled) {
        Event e;
        Read(e);
        if (e.eventType == KeyEvent && e.len > 0) {
            switch (e.keystroke[0]) {
                case 'y':
                case 'Y':
                    yes->SetValue(true);
                    break;
                case 'n':
                case 'N':
                    no->SetValue(true);
                    break;
                case '\r': /* CR */
                case '\007': /* ^G */
                    cancel->SetValue(true);
                    break;
                default:
                    break;
            }
        } else if (e.target == yesbutton || e.target == nobutton ||
            e.target == cancelbutton)
        {
            e.target->Handle(e);
        }
        yes->GetValue(confirmed);
        no->GetValue(denied);
        cancel->GetValue(cancelled);
    }

    Disappear();

    char answer = 'n';
    answer = confirmed ? 'y' : answer;
    answer = cancelled ? 'c' : answer;
    return answer;
}

/*

```

```

    * Init composes Confirmer's view with boxes, glue, and frames.
    */

void Confirmer::Init () {
    SetClassName("Confirmer");

    HBox* buttons = new HBox;
    buttons->Insert(new HGlue);
    buttons->Insert(yesbutton);
    buttons->Insert(new HGlue);
    buttons->Insert(nobutton);
    buttons->Insert(new HGlue);
    buttons->Insert(cancelbutton);
    buttons->Insert(new HGlue);

    VBox* vbox = new VBox;
    vbox->Align(Center);
    vbox->Insert(new VGlue);
    vbox->Insert(warning);
    vbox->Insert(new VGlue);
    vbox->Insert(message);
    vbox->Insert(new VGlue);
    vbox->Insert(buttons);
    vbox->Insert(new VGlue);

    Insert(new Frame(vbox, 2));
}

/*
 * Reconfig pads Confirmer's shape to make the view look less crowded.
 */

void Confirmer::Reconfig () {
    DialogBox::Reconfig();
    Font* font = output->GetFont();
    shape->width += 4 * font->Width("mmmmmm");
    shape->height += 4 * font->Height();
}

/*
 * Namer creates its button states and initializes its view.
 */

Namer::Namer (Interactor* u, const char* prompt) : (u, prompt) {
    accept = new ButtonState(false);
    cancel = new ButtonState(false);
    acceptbutton = new PushButton(" OK ", accept, true);
    cancelbutton = new PushButton("Cancel", cancel, true);
    const char* sample = " ";
    stringeditor = new StringEditor(accept, sample, "\007\015");
    stringeditor->Message("");
    Init();
}

```

```

}

/*
 * Free storage allocated for the button states.
 */

Namer::~Namer () {
    Unref(accept);
    Unref(cancel);
}

/*
 * Edit pops up the Namer, lets the user edit the given string,
 * removes the Namer, and returns the edited string unless the user
 * cancelled it.
 */

char* Namer::Edit (const char* string) {
    accept->SetValue(false);
    cancel->SetValue(false);
    if (string != nil) {
        stringeditor->Message(string);
    }
    stringeditor->Select(0, strlen(stringeditor->Text()));

    PopUp();

    int accepted = false;
    int cancelled = false;
    while (!accepted && !cancelled) {
        stringeditor->Edit();
        accept->GetValue(accepted);
        if (accepted == '\007') {
            accept->SetValue(false);
            cancel->SetValue(true);
        } else if (accepted == '\015') {
            accept->SetValue(true);
            cancel->SetValue(false);
        } else {
            Event e;
            Read(e);
            if (e.target == acceptbutton || e.target == cancelbutton) {
                e.target->Handle(e);
            }
        }
        accept->GetValue(accepted);
        cancel->GetValue(cancelled);
    }

    Disappear();

    char* result = nil;

```



```

        if (accepted) {
            const char* text = stringeditor->Text();
            if (text[0] != '\0') {
                result = strdup(text);
            }
        }
        return result;
    }

/*
 * Init composes Namer's view with boxes, glue, and frames.
 */

void Namer::Init () {
    SetClassName("Namer");

    HBox* hboxedit = new HBox;
    hboxedit->Insert(new HGlue(5, 0, 0));
    hboxedit->Insert(stringeditor);
    hboxedit->Insert(new HGlue(5, 0, 0));

    VBox* vboxedit = new VBox;
    vboxedit->Insert(new VGlue(2, 0, 0));
    vboxedit->Insert(hboxedit);
    vboxedit->Insert(new VGlue(2, 0, 0));

    HBox* buttons = new HBox;
    buttons->Insert(new HGlue);
    buttons->Insert(acceptbutton);
    buttons->Insert(new HGlue);
    buttons->Insert(cancelbutton);
    buttons->Insert(new HGlue);

    VBox* vbox = new VBox;
    vbox->Align(Center);
    vbox->Insert(new VGlue);
    vbox->Insert(warning);
    vbox->Insert(new VGlue);
    vbox->Insert(message);
    vbox->Insert(new VGlue);
    vbox->Insert(new Frame(vboxedit, 1));
    vbox->Insert(new VGlue);
    vbox->Insert(buttons);
    vbox->Insert(new VGlue);
    vbox->Propagate(false); /* for reshaping stringeditor w/o looping */

    Insert(new Frame(vbox, 2));
}

/*
 * Reconfig pads Namer's shape to make the view look less crowded.
 */

```

```

void Namer::Reconfig () {
    DialogBox::Reconfig();
    Shape s = *stringeditor->GetShape();
    s.Rigid();
    stringeditor->Reshape(s);
    Font* font = output->GetFont();
    shape->width += 2 * font->Width("mmmmmm");
    shape->height += 4 * font->Height();
}

static const char* abspath (const char* file = nil) {
    const int bufsize = MAXPATHLEN+1;
    static char buf[bufsize];

    getcwd(buf, bufsize);
    strcat(buf, "/");

    if (file != nil) {
        strcat(buf, file);
    }
    return buf;
}

Finder::Finder (
    Interactor* u, const char* t
) : (new ButtonState, abspath(), 10, 24, Center) {
    underlying = u;
    Init("", t);
    Insert(Interior());
}

static const char* ChdirIfNecessary (Finder* finder) {
    static char buf[MAXPATHLEN+1];
    const char* filename = finder->Choice();
    strcpy(buf, filename);
    char* bufptr = strrchr(buf, '/');

    if (bufptr != NULL) {
        *bufptr = '\0';
        if (chdir(buf) == 0) {
            filename = ++bufptr;
            finder->Message(abspath(filename));
        }
    }
    finder->SelectFile();
    return filename;
}

const char* Finder::Find () {
    const char* name = nil;
    Event e;

```

```

        if (Popup(e)) {
            name = ChdirIfNecessary(this);
        }
        return name;
    }

    Interactor* Finder::Interior () {
        return new Frame(FileChooser::Interior(" Open "), 2);
    }

    boolean Finder::Popup (Event&, boolean) {
        World* world = underlying->GetWorld();
        Coord x, y;
        underlying->Align(Center, 0, 0, x, y);
        underlying->GetRelative(x, y, world);
        world->InsertTransient(this, underlying, x, y, Center);
        boolean accepted = Accept();
        world->Remove(this);
        SetTitle("");
        return accepted;
    }

    /*
    * Chooser creates its button states and initializes its view.
    */

    Chooser::Chooser (Interactor* u, const char* prompt, const char* label_1,
                      const char* label_2, const char* label_3) : (u, prompt) {
        bs_1      = new ButtonState(false);
        bs_2      = new ButtonState(false);
        bs_3      = new ButtonState(false);
        cancel    = new ButtonState(false);
        button_1  = new PushButton(label_1, bs_1, true);
        button_2  = new PushButton(label_2, bs_2, true);
        button_3  = new PushButton(label_3, bs_3, true);
        cancelbutton = new PushButton("    Cancel    ", cancel, true);
        Init();
    }

    /*
    * Free storage allocated for the button states.
    */

    Chooser::~Chooser () {
        Unref(bs_1);
        Unref(bs_2);
        Unref(bs_3);
        Unref(cancel);
    }

    /*
    * Choos pops up the Chooser, lets the user choose one of the buttons,

```

```

    * removes the Chooser, and returns the choice.
    */

char Chooser::Choose () {
    bs_1->SetValue(false);
    bs_2->SetValue(false);
    bs_3->SetValue(false);
    cancel->SetValue(false);

    PopUp();

    int first = false;
    int second = false;
    int third = false;
    int cancelled = false;
    while (!first && !second && !third && !cancelled) {
        Event e;
        Read(e);
        if (e.target == button_1 || e.target == button_2 ||
            e.target == button_3 || e.target == cancelbutton) {
            e.target->Handle(e);
        }
        bs_1->GetValue(first);
        bs_2->GetValue(second);
        bs_3->GetValue(third);
        cancel->GetValue(cancelled);
    }

    Disappear();

    char answer = 'c';
    answer = first ? 'f' : answer;
    answer = second ? 's' : answer;
    answer = third ? 't' : answer;
    return answer;
}

/*
 * Init composes Chooser's view with boxes, glue, and frames.
 */

void Chooser::Init () {
    SetClassName("Chooser");

    HBox* buttons = new HBox;
    buttons->Insert(new HGlue);
    buttons->Insert(button_1);
    buttons->Insert(new HGlue);
    buttons->Insert(button_2);
    buttons->Insert(new HGlue);
    buttons->Insert(button_3);
    buttons->Insert(new HGlue);

```

```

    buttons->Insert(cancelbutton);
    buttons->Insert(new HGlue);

    VBox* vbox = new VBox;
    vbox->Align(Center);
    vbox->Insert(new VGlue);
    vbox->Insert(warning);
    vbox->Insert(new VGlue);
    vbox->Insert(message);
    vbox->Insert(new VGlue);
    vbox->Insert(buttons);
    vbox->Insert(new VGlue);

    Insert(new Frame(vbox, 2));
}

/*
 * Reconfig pads Chooser's shape to make the view look less crowded.
 */

void Chooser::Reconfig () {
    DialogBox::Reconfig();
    Font* font = output->GetFont();
    shape->width += 4 * font->Width("mmmmmm");
    shape->height += 4 * font->Height();
}

```

```

// file          drawing.h
// description:   Class description of Drawing class.

// $Header: drawing.h,v 1.13 89/10/09 14:47:50 linton Exp $
// declares class Drawing.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add 3 TextBuffers to store different PSDL for drawing.
 * Add declaration of DFD specific functions to store the internal
 * DFD representation of the drawing.
 * Add OperatorSelList to store DFD objects and how they are related.
 *
 * Changes made by:   Mary Ann Cummins
 * Last change made:  October 17, 1990
 */

#ifndef drawing_h
#define drawing_h

#include <InterViews/Std/stdio.h>
#include <InterViews/defs.h>
#include <InterViews/Graphic/classes.h>

// Declare imported types.

class BSplineSelection;
class CenterList;
class DrawingView;
class EdgeList;
class EllipseSelection;
class Graphic;
class GroupList;
class IBrush;
class IBrushList;
class IColor;
class IColorList;
class IFont;
class IFontList;
class IPattern;
class IPatternList;
// class LineSelection;
class Page;
class PictSelection;
class Selection;
class SelectionList;
class State;
class Transformer;
class TextBuffer;
class booleanList;
class OperatorSelList;
class TextSelection;

```

```
// A Drawing contains the user's picture and provides the interface
// through which editing operations modify it.
```

```
class Drawing {
public:
```

```
    Drawing(double w, double h, double b);
    ~Drawing();
```

```
    boolean GetLandscape();
    Page* GetPage();
    void GetPictureTT(Transformer&);
    SelectionList* GetSelectionList();
```

```
    boolean Writable(const char*);
    boolean Exists(const char*);
    void ClearPicture();
    boolean ReadPicture(const char*, State*);
    boolean PrintPicture(const char*, State*);
    boolean WritePicture(const char*, State*);
    SelectionList* ReadClipboard(State*);
    void WriteClipboard();
```

```
    void GetBox(Coord&, Coord&, Coord&, Coord&);
    IBrushList* GetBrush();
    CenterList* GetCenter();
    GroupList* GetChildren();
    IColorList* GetFgColor();
    IColorList* GetBgColor();
    SelectionList* GetDuplicates();
    booleanList* GetFillBg();
    IFontList* GetFont();
    int GetNumberOfGraphics();
    GroupList* GetParent();
    IPatternList* GetPattern();
    SelectionList* GetPrevs();
    SelectionList* GetSelections();
```

```
    Selection* PickSelectionIntersecting(Coord, Coord);
    Selection* PickSelectionShapedBy(Coord, Coord);
    SelectionList* PickSelectionsWithin(Coord, Coord, Coord, Coord);
```

```
    void Clear();
    void Extend(Selection*);
    void Extend(SelectionList*);
    void Grasp(Selection*);
    void Grasp(SelectionList*);
    void Select(Selection*);
    void Select(SelectionList*);
    void SelectAll();
```

```

void Move(float, float, DrawingView*);
void Scale(float, float);
void Stretch(float, Alignment);
void Rotate(float);
void Align(Alignment, Alignment);
void AlignToGrid();

void SetBrush(IBrush*);
void SetBrush(IBrushList*);
void SetCenter(CenterList*);
void SetFgColor(IColor*);
void SetFgColor(IColorList*);
void SetBgColor(IColor*);
void SetBgColor(IColorList*);
void SetFillBg(boolean);
void SetFillBg(booleanList*);
void SetFont(IFont*);
void SetFont(IFontList*);
void SetPattern(IPattern*);
void SetPattern(IPatternList*);

void Append();
void Group(GroupList*);
void InsertAfterPrev(SelectionList*);
void Prepend();
void Remove();
void Replace(Selection*, Selection*);
void Sort();
void Ungroup(GroupList*);

// DFD specific functions

EllipseSelection* SetEndptsInOperator(Coord&, Coord&,
                                     Coord&, Coord&);
void OperatorAppend(EllipseSelection*);
// void DataFlowLineAppend(LineSelection*, EllipseSelection*,
//                           EllipseSelection*);
void DataFlowSplineAppend(BSplineSelection*, EllipseSelection*,
                           EllipseSelection*);

void LabelAppend(TextSelection*, Selection*);
void LabelReadAppend(TextSelection*, Selection*);
void AddTextToSelectionList();
void AddSelfLoopsToSelectionList();
void ReplaceAssociatedObjects(EllipseSelection*, DrawingView*);
void WritePSDLForOperator(EllipseSelection*);
void METAppend(TextSelection*, EllipseSelection*);
void WritePSDLForDrawing();
void UpdatePSDLSpec(const char*);
EdgeList* WritePSDLGraph(FILE*);
EdgeList* WriteEdges(FILE*);
EdgeList* BuildEdgeList();
void WriteVertices(FILE*);

```



```

void WriteDFDFiles(char*, const char*);
void WritePSDLForAllOperators(char*, const char*);
void WriteStreams();
void WriteConstraints();
void ReadPSDLForOperator(FILE*, EllipseSelection*);
char* ReadBackPSDL(FILE*);
void ReadPSDLForDrawing();
void ReadStreams();
void ReadConstraints();
void WriteDFDInfo(FILE*, EdgeList*);
void ReadDFDFiles(char*, const char*);
void ReadDFDInfo(FILE*);
char* OperatorLabelIs(EllipseSelection*);
void AddStream();
void LatencyAppend(TextSelection*, BSplineSelection*);
void AddLabelToStreams(char*, TextSelection*);
Selection* FindOperator(TextSelection*);
Selection* FindLabel(Selection*);
void RemoveStream(TextSelection*);
void RemoveAssociatedObjects(SelectionList*, EllipseSelection*);
void RemoveSplines(SelectionList*);

```

protected:

```

    int NumberOfGraphics(PictSelection*);

// DFD specific functions

EllipseSelection* EndptsInOperator(Coord&, Coord&);
void FindOperatorIntersection(Coord&, Coord&, Coord, Coord);
// void ReplaceInputDFLines(Coord, Coord, DrawingView*);
// void ReplaceOutputDFLines(Coord, Coord, DrawingView*);
void ReplaceInputDFSplines(Coord, Coord, DrawingView*);
void ReplaceOutputDFSplines(Coord, Coord, DrawingView*);
void ReplaceSelfLoops(Coord, Coord, DrawingView*);
void ReplaceLabel(Selection*, TextSelection*);
void ReplaceLatency(Selection*, TextSelection*);
void WriteOperator(FILE*, int);
void WriteOpLabel(FILE*, int, int);
void WriteMET(FILE*, int, int);
void WriteSelfLoop(FILE*, int, int);
void WriteSelfLoopLabel(FILE*, int, int, int);
void WriteFlow(FILE*, int, int, int, boolean);
void WriteFlowLabel(FILE*, int, int, int, int);
void WriteFlowLatency(FILE*, int, int, int, int);
void ReadPSDLForAllOperators(char*, const char*);
void FillImpBuffers(char*);
void FindTxbIndices(TextBuffer*, char**, int, int&,
                    char**, int, int&);

char* clipfilename;      // filename under which to store clippings
Page* page;             // draws picture
PictSelection* picture;  // stores picture

```

```
SelectionList* sl;          // lists picked Selections
OperatorSelList* ol;        // list of operators drawn
TextBuffer* spec_txb;       // contains PSDL specification of drawing
TextBuffer* streams_txb;    // contains PSDL streams
TextBuffer* constraints_txb; // contains PSDL constraints of drawing
};

#endif
```

```

// file          drawing.c
// description:   Implementation of Drawing class.

// $Header: drawing.c,v 1.18 89/10/09 14:47:48 linton Exp $
// implements class Drawing.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add functions to manipulate TextBuffers to store specification of
 * drawing, PSDL streams for drawing, and PSDL constraints for drawing.
 * Add overloaded function Grasp to grasp an entire list of selections.
 * Change Move to move operator's related objects when moving operator.
 * Add internal representation of DFD by adding functions to manipulate
 * the operator list.
 * Add ability to write file to rebuild operator list data struction if
 * entering editor with existing drawing.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 17, 1990
 */

#include "dfd_defs.h"
#include "dfdclasses.h"
#include "dfdsplinelist.h"
#include "drawingview.h"
#include "drawing.h"
#include "edge.h"
#include "edgelist.h"
#include "ipaint.h"
#include "istring.h"
#include "listboolean.h"
#include "listcenter.h"
#include "listchange.h"
#include "listgroup.h"
#include "listibrush.h"
#include "listicolor.h"
#include "listifont.h"
#include "listipattern.h"
#include "listselectn.h"
#include "opsellist.h"
#include "page.h"
#include "sldfdspline.h"
#include "sllellipses.h"
#include "sloperator.h"
#include "slpict.h"
#include "slsplines.h"
#include "slttext.h"
#include <InterViews/Graphic/polygons.h>
#include <InterViews/defs.h>
#include <InterViews/regexp.h>
#include <InterViews/textbuffer.h>
#include <InterViews/transformer.h>

```

```

#include <InterViews/Std/os/fs.h>
#include <InterViews/Std/stdio.h>
#include <stdlib.h>
#include <sys/file.h>      /* define constants for access call */
#include <math.h>

// Drawing creates the page, selection list, and clipboard filename.

Drawing::Drawing (double w, double h, double b) {
    const char* home = (home = getenv("HOME")) ? home : ".";
    const char* name = ".clipboard";
    clipfilename = new char[strlen(home) + 1 + strlen(name) + 1];
    strcpy(clipfilename, home);
    strcat(clipfilename, "/");
    strcat(clipfilename, name);
    page = new Page(w, h, b);
    picture = page->GetPicture();
    sl = new SelectionList;
    ol = new OperatorSelList;

    // build initial PSDL specification to put in text buffer

    char* spec_string = new char[TEXTBUFLLEN];
    strcpy(spec_string, OPER_TKN);
    strcat(spec_string, ID_TKN);
    strcat(spec_string, "\n");
    strcat(spec_string, SPEC_TKN);
    strcat(spec_string, DESC_TKN);
    strcat(spec_string, TEXT_TKN);
    strcat(spec_string, END_TKN);
    spec_txb = new TextBuffer(spec_string, strlen(spec_string),
                               TEXTBUFLLEN);

    streams_txb = nil;
    constraints_txb = nil;
}

// ~Drawing frees storage allocated for the clipboard filename, page,
// and selection list.

Drawing::~~Drawing () {
    delete clipfilename;
    delete page;
    delete sl;
    delete ol;
}

// Define access functions to return attributes.

boolean Drawing::GetLandscape () {
    Transformer* t = page->GetTransformer();
    return t ? t->Rotated90() : false;
}

```

```

Page* Drawing::GetPage () {
    return page;
}

void Drawing::GetPictureTT (Transformer& t) {
    picture->TotalTransformation(t);
}

SelectionList* Drawing::GetSelectionList () {
    return sl;
}

// Writable returns true only if the given drawing is writable.

boolean Drawing::Writable (const char* path) {
    return (access(path, W_OK) >= 0);
}

// Exists returns true only if the drawing already exists.

boolean Drawing::Exists (const char* path) {
    return (access(path, F_OK) >= 0);
}

// ClearPicture deletes the old picture and creates a new empty
// picture.

void Drawing::ClearPicture () {
    sl->DeleteAll();
    page->SetPicture(nil);
    picture = page->GetPicture();
}

// ReadPicture reads a new picture and replaces the old picture with
// the new picture if the read succeeds.

boolean Drawing::ReadPicture (const char* path, State* state) {
    boolean successful = false;
    if (path != nil) {
        FILE* stream = fopen(path, "r");
        if (stream != nil) {
            PictSelection* newpic = new PictSelection(stream, state);
            fclose(stream);
            if (newpic->Valid()) {
                sl->DeleteAll();
                page->SetPicture(newpic);
                picture = page->GetPicture();
                successful = true;
            } else {
                delete newpic;
                fprintf(stderr, "Drawing: input error in reading %s\n", path);
            }
        }
    }
}

```

```

        }
    }
    return successful;
}

// PrintPicture prints the current picture by writing it through a
// pipe to a print command.

boolean Drawing::PrintPicture (const char* cmd, State* state) {
    boolean successful = false;
    if (cmd != nil) {
        FILE* stream = popen(cmd, "w");
        if (stream != nil) {
            successful = picture->WritePicture(stream, state, true);
            pclose(stream);
        }
    }
    return successful;
}

// WritePicture writes the current picture to a file.

boolean Drawing::WritePicture (const char* path, State* state) {
    boolean successful = false;
    if (path != nil) {
        FILE* stream = fopen(path, "w");
        if (stream != nil) {
            successful = picture->WritePicture(stream, state, true);
            fclose(stream);
        }
    }
    return successful;
}

// ReadClipboard returns copies of the Selections within the clipboard
// file in a newly allocated list.

SelectionList* Drawing::ReadClipboard (State* state) {
    SelectionList* sl = new SelectionList;
    FILE* stream = fopen(cliptfilename, "r");
    if (stream != nil) {
        PictSelection* newpic = new PictSelection(stream, state);
        fclose(stream);
        if (newpic->Valid()) {
            newpic->Propagate();
            for (newpic->First(); !newpic->AtEnd(); newpic->RemoveCur()) {
                Selection* child = (Selection*) newpic->GetCurrent();
                sl->Append(new SelectionNode(child));
            }
        }
        delete newpic;
    }
}

```

```

    } else {
        fprintf(stderr, "Drawing: can't open %s\n", clipfilename);
    }
    return sl;
}

// WriteClipboard writes the picked Selections to the clipboard file,
// overwriting its previous contents.

void Drawing::WriteClipboard () {
    FILE* stream = fopen(clipfilename, "w");
    if (stream != nil) {
        PictSelection* newpic = new PictSelection;
        for (sl->First(); !sl->AtEnd(); sl->Next()) {
            Graphic* copy = sl->GetCur()->GetSelection()->Copy();
            newpic->Append(copy);
        }
        newpic->WritePicture(stream, nil, false);
        fclose(stream);
        delete newpic;
    } else {
        fprintf(stderr, "Drawing: can't open %s\n", clipfilename);
    }
}

// GetBox gets the smallest box bounding all the Selections.

void Drawing::GetBox (Coord& l, Coord& b, Coord& r, Coord& t) {
    BoxObj btotat;
    BoxObj bselection;

    if (sl->Size() >= 1) {
        sl->First()->GetSelection()->GetBox(btotat);
        for (sl->Next(); !sl->AtEnd(); sl->Next()) {
            sl->GetCur()->GetSelection()->GetBox(bselection);
            btotat = btotat + bselection;
        }
        l = btotat.left;
        b = btotat.bottom;
        r = btotat.right;
        t = btotat.top;
    }
}

// GetBrush returns the Selections' brush attributes in a newly
// allocated list.

IBrushList* Drawing::GetBrush () {
    IBrushList* brushlist = new IBrushList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        IBrush* brush = (IBrush*) sl->GetCur()->GetSelection()->GetBrush();
        brushlist->Append(new IBrushNode(brush));
    }
}

```

```

    }
    return brushlist;
}

// GetCenter returns the Selections' centers in a newly allocated
// list. It converts the centers from window coordinates to picture
// coordinates because only these coordinates will remain constant.

CenterList* Drawing::GetCenter () {
    CenterList* centerlist = new CenterList;
    Transformer t;
    picture->TotalTransformation(t);
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        float wincx, wincy, cx, cy;

        sl->GetCur()->GetSelection()->GetCenter(wincx, wincy);
        t.InvTransform(wincx, wincy, cx, cy);
        centerlist->Append(new CenterNode(cx, cy));
    }
    return centerlist;
}

// GetChildren returns the Selections and their children, if any, in a
// newly allocated list.

GroupList* Drawing::GetChildren () {
    GroupList* grouplist = new GroupList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        PictSelection* parent = (PictSelection*) sl->GetCur()->GetSelection();
        boolean haschildren = parent->HasChildren();
        SelectionList* children = new SelectionList;
        if (haschildren) {
            for (parent->First(); !parent->AtEnd(); parent->Next()) {
                Selection* child = parent->GetCurrent();
                children->Append(new SelectionNode(child));
            }
        }
        grouplist->Append(new GroupNode(parent, haschildren, children));
        delete children;
    }
    return grouplist;
}

// GetFgColor returns the Selections' FgColor attributes in a newly
// allocated list.

IColorList* Drawing::GetFgColor () {
    IColorList* fgcolorlist = new IColorList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        IColor* fgcolor = (IColor*) sl->GetCur()->GetSelection()->GetFgColor();
        fgcolorlist->Append(new IColorNode(fgcolor));
    }
}

```



```

        return fgcolorlist;
    }

// GetBgColor returns the Selections' BgColor attributes in a newly
// allocated list.

IColorList* Drawing::GetBgColor () {
    IColorList* bgcolorlist = new IColorList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        IColor* bgcolor = (IColor*) sl->GetCur()->GetSelection()->
            GetBgColor();
        bgcolorlist->Append(new IColorNode(bgcolor));
    }
    return bgcolorlist;
}

// GetDuplicates duplicates the Selections, offsets them by one grid
// spacing, and returns them in a newly allocated list.

SelectionList* Drawing::GetDuplicates () {
    int offset = round(page->GetGridSpacing() * points);
    SelectionList* duplicates = new SelectionList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* dup = (Selection*) sl->GetCur()->GetSelection()->Copy();
        dup->Translate(offset, offset);
        duplicates->Append(new SelectionNode(dup));
    }
    return duplicates;
}

// GetFillBg returns the Selections' fillbg attributes in a newly
// allocated list.

booleanList* Drawing::GetFillBg () {
    booleanList* fillbglist = new booleanList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        boolean fillbg = sl->GetCur()->GetSelection()->BgFilled();
        fillbglist->Append(new booleanNode(fillbg));
    }
    return fillbglist;
}

// GetFont returns the Selections' Font attributes in a newly
// allocated list.

IFontList* Drawing::GetFont () {
    IFontList* fontlist = new IFontList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        IFont* font = (IFont*) sl->GetCur()->GetSelection()->GetFont();
        fontlist->Append(new IFontNode(font));
    }
    return fontlist;
}

```

```

}

// GetNumberOfGraphics returns the number of graphics in the
// Selections.

int Drawing::GetNumberOfGraphics () {
    int num = 0;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        if (s->HasChildren()) {
            num += NumberOfGraphics((PictSelection*) s);
        } else {
            ++num;
        }
    }
    return num;
}

// GetParent returns the Selections and their new parent in a newly
// allocated list if there are enough Selections to form a Group.

GroupList* Drawing::GetParent () {
    GroupList* grouplist = new GroupList;
    if (sl->Size() >= 2) {
        PictSelection* parent = new PictSelection;
        boolean haschildren = true;
        grouplist->Append(new GroupNode(parent, haschildren, sl));
    }
    return grouplist;
}

// GetPattern returns the Selections' Pattern attributes in a newly
// allocated list.

IPatternList* Drawing::GetPattern () {
    IPatternList* patternlist = new IPatternList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        IPattern* pattern =
            (IPattern*) sl->GetCur()->GetSelection()->GetPattern();
        patternlist->Append(new IPatternNode(pattern));
    }
    return patternlist;
}

// GetPrevs returns the Selections' predecessors within the picture in
// a newly allocated list.

SelectionList* Drawing::GetPrevs () {
    SelectionList* prevlist = new SelectionList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        picture->SetCurrent(sl->GetCur()->GetSelection());
        Selection* prev = picture->Prev();
    }
}

```

```

    prevlist->Append(new SelectionNode(prev));
    }
    return prevlist;
}

// GetSelections returns the Selections in a newly allocated list.

SelectionList* Drawing::GetSelections () {
    SelectionList* newsl = new SelectionList;
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        newsl->Append(new SelectionNode(s));
    }
    return newsl;
}

// PickSelectionIntersecting returns the last Selection intersecting a
// box around the given point.

Selection* Drawing::PickSelectionIntersecting (Coord x, Coord y) {
    const int SLOP = 2;
    BoxObj pickpoint(x - SLOP, y - SLOP, x + SLOP, y + SLOP);
    return picture->LastSelectionIntersecting(pickpoint);
}

// PickSelectionShapedBy returns the last Selection shaped by a point
// close to the given point.

Selection* Drawing::PickSelectionShapedBy (Coord x, Coord y) {
    const float SLOP = 6.;
    for (picture->Last(); !picture->AtEnd(); picture->Prev()) {
        Selection* pick = picture->GetCurrent();
        if (pick->ShapedBy(x, y, SLOP)) {
            return pick;
        }
    }
    return nil;
}

// PickSelectionsWithin returns all the Selections within the given
// box.

SelectionList* Drawing::PickSelectionsWithin (Coord l, Coord b, Coord r,
Coord t) {
    Selection** picks = nil;
    int numpicks = picture->SelectionsWithin(BoxObj(l, b, r, t), picks);
    SelectionList* picklist = new SelectionList;
    for (int i = 0; i < numpicks; i++) {
        if (!picklist->Find(picks[i])) {
            picklist->Append(new SelectionNode(picks[i]));
        }
    }
}

```

```

        delete picks;
        return picklist;
    }

    // Clear empties the SelectionList.

    void Drawing::Clear () {
        sl->DeleteAll();
    }

    // Extend extends the SelectionList to include the picked Selection
    // unless it's already there, in which case it removes the Selection.

    void Drawing::Extend (Selection* pick) {
        if (!sl->Find(pick)) {
            sl->Append(new SelectionNode(pick));
        }
        else {
            sl->DeleteCur();
        }
    }

    // Extend extends the SelectionList to include the picked Selections
    // unless they're already there, in which case it removes them.

    void Drawing::Extend (SelectionList* picklist) {
        for (picklist->First(); !picklist->AtEnd(); picklist->Next()) {
            Selection* pick = picklist->GetCur()->GetSelection();
            Extend(pick);
        }
    }

    // Grasp selects the picked Selection only if the SelectionList does
    // not already include it.

    void Drawing::Grasp (Selection* pick) {
        if (!sl->Find(pick)) {
            Select(pick);
        }
    }

    // Grasp selects the list of picked Selections only if the SelectionList
    // does not already include any of the selections.

    void Drawing::Grasp (SelectionList* picklist) {
        boolean found = false;
        for (picklist->First(); !picklist->AtEnd(); picklist->Next()) {
            if (sl->Find(picklist->GetCur()->GetSelection())) {
                found = true;
                exit;
            }
        }
    }

```

```

        if (!found) {
            Select(picklist);
        }
    }

    // Select selects the picked Selection.

void Drawing::Select (Selection* pick) {
    sl->DeleteAll();
    sl->Append(new SelectionNode(pick));
}

    // Select selects the picked Selections.

void Drawing::Select (SelectionList* picklist) {
    sl->DeleteAll();
    for (picklist->First(); !picklist->AtEnd(); picklist->Next()) {
        Selection* pick = picklist->GetCur()->GetSelection();
        sl->Append(new SelectionNode(pick));
    }
}

    // SelectAll selects all of the Selections in the picture.

void Drawing::SelectAll () {
    sl->DeleteAll();
    for (picture->First(); !picture->AtEnd(); picture->Next()) {
        Selection* pick = picture->GetCurrent();
        sl->Append(new SelectionNode(pick));
    }
}

    // Move translates the Selections.

void Drawing::Move (float xdisp, float ydisp, DrawingView* dv) {

    // add labels, mets, and selfloops and their labels to the list of
    // selections
    // to be moved if the operator they are attached to is to be moved.
    // These objects can be translated along with the operator - data flows
    // cannot be translated, they must be modified

    AddTextToSelectionList();
    AddSelfLoopsToSelectionList();
    ClassId cid;
    int sl_size = sl->Size();
    for (int index = 0; index < sl_size; ++index) {
        Selection* s = sl->Index(index)->GetSelection();
        cid = s->GetClassId();
        if (cid == OPERATOR || cid == COMMENT ||
            cid == LABEL_OP || cid == MET_OP ||
            cid == LABEL_SL || cid == SELFLOOP || (sl_size == 1 &&

```

```

        (cid == LABEL_DF || cid == LAT_DF))) {
        s->Translate(xdisp, ydisp);
        if (cid == OPERATOR) {
            SelectionList* temp = GetSelections();
            ReplaceAssociatedObjects((EllipseSelection*) s, dv);
            Select(temp);
        }
    }
}

// only redraw the screen once to make the transition look smooth

dv->Draw();
}

// Scale scales the Selections about their centers.

void Drawing::Scale (float xscale, float yscale) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        float cx, cy;
        s->GetCenter(cx, cy);
        s->Scale(xscale, yscale, cx, cy);
    }
}

// Stretch stretches the Selections while keeping the given side
// fixed.

void Drawing::Stretch (float stretch, Alignment side) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        float l, b, r, t;
        s->GetBounds(l, b, r, t);
        switch (side) {
            case Left:
                s->Scale(stretch, l, r, t);
                break;
            case Bottom:
                s->Scale(l, stretch, r, t);
                break;
            case Right:
                s->Scale(stretch, l, l, b);
                break;
            case Top:
                s->Scale(l, stretch, l, b);
                break;
            default:
                fprintf(stderr, "inappropriate enum passed to Drawing::Stretch\n");
                break;
        }
    }
}

```

```

}

// Rotate rotates the Selections about their centers.

void Drawing::Rotate (float angle) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        float cx, cy;
        s->GetCenter(cx, cy);
        s->Rotate(angle, cx, cy);
    }
}

// Align either aligns up all of the Selections or abuts all of them
// side to side, depending on whether the moving Selection's side or
// center aligns with the fixed Selection's same side or center.

void Drawing::Align (Alignment falign, Alignment malign) {
    if (falign == malign) {
        Selection* stays = sl->First()->GetSelection();
        for (sl->Next(); !sl->AtEnd(); sl->Next()) {
            Selection* moves = sl->GetCur()->GetSelection();
            stays->Align(falign, moves, malign);
        }
    } else {
        Selection* stays = sl->First()->GetSelection();
        for (sl->Next(); !sl->AtEnd(); sl->Next()) {
            Selection* moves = sl->GetCur()->GetSelection();
            stays->Align(falign, moves, malign);
            stays = moves;
        }
    }
}

// AlignToGrid aligns the Selections' lower left corners to the
// nearest grid point.

void Drawing::AlignToGrid () {
    boolean gravity = page->GetGridGravity();
    page->SetGridGravity(true);
    Transformer t;
    picture->TotalTransformation(t);

    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        float l, b, dummy;
        s->GetBounds(l, b, dummy, dummy);
        Coord nl = round(l);
        Coord nb = round(b);
        page->Constrain(nl, nb);
        float x0, y0, x1, y1;
        t.InvTransform(l, b, x0, y0);
    }
}

```

```

        t.InvTransform(float(nl), float(nb), x1, y1);
        s->Translate(x1 - x0, y1 - y0);
    }

    page->SetGridGravity(gravity);
}

// SetBrush sets the Selections' brush attributes with the given brush
// attribute.

void Drawing::SetBrush (IBrush* brush) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        sl->GetCur()->GetSelection()->SetBrush(brush);
    }
}

// SetBrush sets each Selection's brush attribute with the
// corresponding brush attribute in the provided list.

void Drawing::SetBrush (IBrushList* brushlist) {
    for (sl->First(), brushlist->First();
         !sl->AtEnd() && !brushlist->AtEnd();
         sl->Next(), brushlist->Next())
    {
        IBrush* brush = brushlist->GetCur()->GetBrush();
        sl->GetCur()->GetSelection()->SetBrush(brush);
    }
}

// SetCenter centers each of the Selections over the corresponding
// position in the provided list. It expects the passed positions to
// in picture coordinates, not window coordinates.

void Drawing::SetCenter (CenterList* centerlist) {
    Transformer t;
    picture->TotalTransformation(t);
    for (sl->First(), centerlist->First();
         !sl->AtEnd() && !centerlist->AtEnd();
         sl->Next(), centerlist->Next())
    {
        float winoldcx, winoldcy, oldcx, oldcy;
        float newcx = centerlist->GetCur()->GetCx();
        float newcy = centerlist->GetCur()->GetCy();
        Selection* s = sl->GetCur()->GetSelection();

        s->GetCenter(winoldcx, winoldcy);
        t.InvTransform(winoldcx, winoldcy, oldcx, oldcy);
        s->Translate(newcx - oldcx, newcy - oldcy);
    }
}

// SetFgColor sets the Selections' foreground color attributes with

```



```

// the given color attribute.

void Drawing::SetFgColor (IColor* fgcolor) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        IColor* bgcolor = (IColor*) s->GetBgColor();
        s->SetColors(fgcolor, bgcolor);
    }
}

// SetFgColor sets the Selections' foreground color attributes with
// the corresponding color attributes in the provided list.

void Drawing::SetFgColor (IColorList* fgcolorlist) {
    for (sl->First(), fgcolorlist->First();
        !sl->AtEnd() && !fgcolorlist->AtEnd();
        sl->Next(), fgcolorlist->Next())
    {
        Selection* s = sl->GetCur()->GetSelection();
        IColor* fgcolor = fgcolorlist->GetCur()->GetColor();
        IColor* bgcolor = (IColor*) s->GetBgColor();
        s->SetColors(fgcolor, bgcolor);
    }
}

// SetBgColor sets the Selections' background color attributes with
// the given color attribute.

void Drawing::SetBgColor (IColor* bgcolor) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        IColor* fgcolor = (IColor*) s->GetFgColor();
        s->SetColors(fgcolor, bgcolor);
    }
}

// SetBgColor sets the Selections' background color attributes with
// the corresponding color attributes in the provided list.

void Drawing::SetBgColor (IColorList* bgcolorlist) {
    for (sl->First(), bgcolorlist->First();
        !sl->AtEnd() && !bgcolorlist->AtEnd();
        sl->Next(), bgcolorlist->Next())
    {
        Selection* s = sl->GetCur()->GetSelection();
        IColor* fgcolor = (IColor*) s->GetFgColor();
        IColor* bgcolor = bgcolorlist->GetCur()->GetColor();
        s->SetColors(fgcolor, bgcolor);
    }
}

// SetFillBg sets the Selections' fillbg attributes with the given

```

```

// fillbg attribute.

void Drawing::SetFillBg (boolean fillbg) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        sl->GetCur()->GetSelection()->FillBg(fillbg);
    }
}

// SetFillBg sets each Selection's fillbg attribute with the
// corresponding fillbg attribute in the provided list.

void Drawing::SetFillBg (booleanList* fillbglist) {
    for (sl->First(), fillbglist->First();
        !sl->AtEnd() && !fillbglist->AtEnd();
        sl->Next(), fillbglist->Next())
    {
        boolean fillbg = fillbglist->GetCur()->GetBoolean();
        sl->GetCur()->GetSelection()->FillBg(fillbg);
    }
}

// SetFont sets the Selections' font attributes with the given font
// attribute.

void Drawing::SetFont (IFont* font) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        s->SetFont(font);
    }
}

// SetFont sets each Selection's font attribute with the corresponding
// font attribute in the provided list.

void Drawing::SetFont (IFontList* fontlist) {
    for (sl->First(), fontlist->First(); !sl->AtEnd() &&
        !fontlist->AtEnd();
        sl->Next(), fontlist->Next())
    {
        IFont* font = fontlist->GetCur()->GetFont();
        Selection* s = sl->GetCur()->GetSelection();
        s->SetFont(font);
    }
}

// SetPattern sets the Selections' pattern attributes with the given
// pattern attribute.

void Drawing::SetPattern (IPattern* pattern) {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        sl->GetCur()->GetSelection()->SetPattern(pattern);
    }
}

```

```

}

// SetPattern sets each Selection's pattern attribute with the
// corresponding pattern attribute in the provided list.

void Drawing::SetPattern (IPatternList* patternlist) {
    for (sl->First(), patternlist->First();
        !sl->AtEnd() && !patternlist->AtEnd();
        sl->Next(), patternlist->Next())
    {
        IPattern* pattern = patternlist->GetCur()->GetPattern();
        sl->GetCur()->GetSelection()->SetPattern(pattern);
    }
}

// Append appends the Selections to the picture.

void Drawing::Append () {
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        picture->Append(s);
    }
}

// Group groups each parent's children, if any, under their parent and
// returns the resulting Selections in the SelectionList.

void Drawing::Group (GroupList* grouplist) {
    if (grouplist->Size() >= 1) {
        sl->DeleteAll();
        for (grouplist->First(); !grouplist->AtEnd(); grouplist->Next()) {
            GroupNode* gn = grouplist->GetCur();
            PictSelection* parent = gn->GetParent();
            boolean haschildren = gn->GetHasChildren();
            SelectionList* children = gn->GetChildren();
            SelectionList* childrengs = gn->GetChildrenGS();
            if (haschildren) {
                for (children->First(), childrengs->First();
                    !children->AtEnd() && !childrengs->AtEnd();
                    children->Next(), childrengs->Next())
                {
                    Graphic* child = children->GetCur()->GetSelection();
                    Graphic* childgs = childrengs->GetCur()->GetSelection();
                    *child = *childgs;
                    picture->SetCurrent(child);
                    picture->Remove(child);
                    parent->Append(child);
                }
            }
            picture->InsertBeforeCur(parent);
            sl->Append(new SelectionNode(parent));
        }
    }
}

```

```

    Sort();
    }
}

// InsertAfterPrev inserts each Selection after its corresponding
// predecessor in the provided list.

void Drawing::InsertAfterPrev (SelectionList* prevlist) {
    for (sl->First(), prevlist->First(); !sl->AtEnd() &&
                                               !prevlist->AtEnd();
        sl->Next(), prevlist->Next())
    {
        Selection* prev = prevlist->GetCur()->GetSelection();
        picture->SetCurrent(prev);
        Selection* s = sl->GetCur()->GetSelection();
        picture->InsertAfterCur(s);
    }
}

// Prepend prepends the Selections to the picture.

void Drawing::Prepend () {
    for (sl->Last(); !sl->AtEnd(); sl->Prev()) {
        Selection* s = sl->GetCur()->GetSelection();
        picture->Prepend(s);
    }
}

// Remove removes the Selections from the picture.

void Drawing::Remove () {
    SelectionList* sl_2 = GetSelections();
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        Selection* s = sl->GetCur()->GetSelection();
        TextSelection* ts1 = nil;
        TextSelection* ts2 = nil;
        ClassId cid = s->GetClassId();
        switch (cid) {
            case OPERATOR:
                ts1 = ol->GetOperatorLabel((EllipseSelection*) s);
                ts2 = ol->GetOperatorMET((EllipseSelection*) s);
                RemoveAssociatedObjects(sl_2, (EllipseSelection*) s);
                break;
            case DATAFLOW_SPLINE:
                ts1 = ol->GetDFLabel((BSplineSelection*) s);
                ts2 = ol->GetDFLatency((BSplineSelection*) s);
                if ((BSplineSelection*) s->IsAStream()) {
                    RemoveStream(ts1);
                }
                break;
            case SELFLOOP:
                ts1 = ol->GetSelfLoopLabel((BSplineSelection*) s);

```

```

        break;
    case LABEL_DF:
        BSplineSelection* flow = ol->GetFlow((TextSelection*) s);
        if (flow != nil && flow->IsAStream()) {
            int len;
            const char* tmp = ((TextSelection*)s)->GetOriginal(len);
            char* old_string = new char[len+1];
            strncpy(old_string,tmp,len);
            old_string[len] = '\0';
            TextSelection* tmp_ts = nil;
            AddLabelToStreams(old_string, (TextSelection*) tmp_ts);
        }
        break;
    default:
        break;
}
ol->Remove(s);
if (ts1 != nil && !sl_2->Find(ts1)) {
    picture->Remove(ts1);
}
if (ts2 != nil && !sl_2->Find(ts2)) {
    picture->Remove(ts2);
}
picture->Remove(s);
}
}

```

// Replace replaces a Selection in the picture with a Selection not in it.

```
void Drawing::Replace (Selection* replacee, Selection* replacer) {
```

// replace objects in operator list too

```

    if (replacee->GetClassId() == LABEL_DF) {
        BSplineSelection* flow = ol->GetFlow((TextSelection*) replacee);
        if (flow != nil && flow->IsAStream()) {
            int len;
            const char* tmp = ((TextSelection*)replacee)->GetOriginal(len);
            char* old_string = new char[len+1];
            strncpy(old_string,tmp,len);
            old_string[len] = '\0';
            AddLabelToStreams(old_string, (TextSelection*) replacer);
        }
    }
    if (replacee->GetClassId() == DATAFLOW_SPLINE &&
        ((BSplineSelection*) replacee)->IsAStream()) {
        ((BSplineSelection*)replacer)->SetStream();
    }
    ol->Replace(replacee, replacer);
    picture->SetCurrent(replacee);
    picture->Remove(replacee);
    picture->InsertBeforeCur(replacer);
}

```

```

}

// Sort sorts the Selections so they occur in the same order as they
// do in the picture.

void Drawing::Sort () {
    if (sl->Size() >= 2) {
        for (picture->First(); !picture->AtEnd(); picture->Next()) {
            Selection* g = picture->GetCurrent();
            if (sl->Find(g)) {
                SelectionNode* s = sl->GetCur();
                sl->RemoveCur();
                sl->Append(s);
            }
        }
    }
}

// Ungroup replaces all Selections which contain children with their
// children and returns the resulting Selections in the SelectionList.

void Drawing::Ungroup (GroupList* grouplist) {
    if (grouplist->Size() >= 1) {
        sl->DeleteAll();
        for (grouplist->First(); !grouplist->AtEnd(); grouplist->Next()) {
            GroupNode* gn = grouplist->GetCur();
            PictSelection* parent = gn->GetParent();
            boolean haschildren = gn->GetHasChildren();
            SelectionList* children = gn->GetChildren();
            if (haschildren) {
                parent->Propagate();
                picture->SetCurrent(parent);
                for (children->First(); !children->AtEnd(); children->Next()) {
                    Selection* child = children->GetCur()->GetSelection();
                    parent->Remove(child);
                    picture->InsertBeforeCur(child);
                    sl->Append(new SelectionNode(child));
                }
                picture->Remove(parent);
            } else {
                sl->Append(new SelectionNode(parent));
            }
        }
        Sort();
    }
}

// NumberOfGraphics returns the number of graphics in the picture,
// calling itself recursively to count the number of graphics in
// subpictures.

int Drawing::NumberOfGraphics (PictSelection* picture) {

```

```

int num = 0;
for (picture->First(); !picture->AtEnd(); picture->Next()) {
    Selection* s = picture->GetCurrent();
    if (s->HasChildren()) {
        num += NumberOfGraphics((PictSelection*) s);
    } else {
        ++num;
    }
}
return num;
}

// DFD specific functions

// return ellipse that contains the given points

EllipseSelection* Drawing::EndptsInOperator(Coord& x, Coord& y) {
    ol->SetCurContaining(x,y);
    if (!ol->AtEnd()) {
        float fx, fy;
        ol->GetCur()->GetSelection()->GetEllipseSelection()
            ->GetCenter(fx, fy);

        x = (Coord) fx;
        y = (Coord) fy;
        return ol->GetCur()->GetSelection()->GetEllipseSelection();
    }
    else {
        return nil;
    }
}

// Find point where data flow intersects with operator in order to
// draw data flow's endpoints at intersection point with operator

void Drawing::FindOperatorIntersection(Coord& x, Coord& y, Coord x2,
                                       Coord y2) {
    if (x != x2 && y != y2) {
        double temp1, temp2, h, a, den;

        temp1 = (double) (x2 - x);
        temp2 = (double) (y2 - y);
        den = sqrt((temp1 * temp1) + (temp2 * temp2));
        a = (double) (OperatorRadius) * (temp1 / den);
        h = (temp2 / temp1) * a;

        x += (Coord) a;
        y += (Coord) h;
    }
    else {
        if (x == x2) {
            if (y2 > y) {
                y += OperatorRadius;
            }
        }
    }
}

```

```

        }
        else {
            if (y2 < y) {
                y -= OperatorRadius;
            }
        }
    }
    else {
        if (y == y2) {
            if (x2 > x) {
                x += OperatorRadius;
            }
            else {
                if (x2 < x) {
                    x -= OperatorRadius;
                }
            }
        }
    }
}

// Finds the operator that contains the given points and modifies
// those points to be where the data flow intersects with the operator

EllipseSelection* Drawing::SetEndptsInOperator(Coord& x1, Coord& y1,
                                                Coord& x2, Coord& y2) {
    EllipseSelection* es1 = EndptsInOperator(x1,y1);

    if (es1 != nil) {
        FindOperatorIntersection(x1, y1, x2, y2);
    }
    return es1;
}

// Add ellipse to operator list

void Drawing::OperatorAppend(EllipseSelection* es) {
    OperatorSelection* os = new OperatorSelection(es);
    ol->Append(new OperatorSelNode(os));
}

// Lines no longer used, use splines instead

/* ***** Start of Commented Out Code *****

void Drawing::DataFlowLineAppend(LineSelection* ls,
                                EllipseSelection* es0,
                                EllipseSelection* es1) {
    DFDLineSelection* DFDLine = new DFDLineSelection(ls);
    if (es0 != nil) {
        ol->SetCur(es0);
        if (!ol->AtEnd()) {

```



```

        ol->GetCur()->GetSelection()->
            OutputDataFlowLineAppend(new DFDLineSelNode(DFDLine));
    }
}

if (es1 != nil) {
    ol->SetCur(es1);
    if (!ol->AtEnd()) {
        ol->GetCur()->GetSelection()->
            InputDataFlowLineAppend(new DFDLineSelNode(DFDLine));
    }
}

}

***** End of Commented Out Code ***** */

// Add data flow to operator list

void Drawing::DataFlowSplineAppend(BSplineSelection* ss,
    EllipseSelection* es0, EllipseSelection* esn) {
    DFDSplineSelection* DFDSpline = new DFDSplineSelection(ss);
    if (es0 == esn) {
        ol->SetCur(es0);
        if (!ol->AtEnd()) {
            ol->GetCur()->GetSelection()->
                SelfLoopAppend(new DFDSplineSelNode(DFDSpline));
        }
    }
    else {
        if (es0 != nil) {
            ol->SetCur(es0);
            if (!ol->AtEnd()) {
                ol->GetCur()->GetSelection()->OutputDataFlowSplineAppend(
                    new DFDSplineSelNode(DFDSpline));
            }
        }

        if (esn != nil) {
            ol->SetCur(esn);
            if (!ol->AtEnd()) {
                ol->GetCur()->GetSelection()->InputDataFlowSplineAppend(
                    new DFDSplineSelNode(DFDLine));
            }
        }
    }
}

// Add label to operator list

void Drawing::LabelAppend(TextSelection* ts, Selection* sel) {
    ClassId classid = sel->GetClassId();
    switch (classid) {
        case OPERATOR:

```

```

        ol->SearchOperators(ts, (EllipseSelection*) sel);
        break;
/* ***** Start of Commented Out Code
case DATAFLOW_LINE:
    ol->SearchOperatorsForLine(ts, (LineSelection*) sel);
    break;
***** End of Commented Out Code ***** */
case DATAFLOW_SPLINE:
    char* old_name =
        ol->SearchOperatorsForSpline(ts, (BSplineSelection*) sel);
    if (old_name != nil) {
        AddLabelToStreams(old_name, ts);
    }
    break;
case SELFLOOP:
    ol->SearchOperatorsForSpline(ts, (BSplineSelection*) sel);
    break;
}
}

// Add label to operator list. When reading from file, do not want to
// update PSDL.

void Drawing::LabelReadAppend(TextSelection* ts, Selection* sel) {
    ClassId classid = sel->GetClassId();
    switch (classid) {
        case OPERATOR:
            ol->SearchOperators(ts, (EllipseSelection*) sel);
            break;
        case DATAFLOW_SPLINE:
            char* old_name =
                ol->SearchOperatorsForSpline(ts, (BSplineSelection*) sel);
            break;
        case SELFLOOP:
            ol->SearchOperatorsForSpline(ts, (BSplineSelection*) sel);
            break;
    }
}

// For all operators selected, add their mets and labels to the selection
// list

void Drawing::AddTextToSelectionList() {
    int op_index = 0, sl_index = 0;
    int op_index_array[sl->Size()];
    for (sl ->First(); !sl->AtEnd(); sl->Next()) {
        if (sl->GetCur()->GetSelection()->GetClassId() == OPERATOR) {
            op_index_array[op_index++] = sl_index;
        }
        ++sl_index;
    }
}

```

```

Selection* s;
TextSelection* ts;
TextSelection* met_ts;
TextSelection* lat_ts;
for (int i = 0; i < op_index; ++i) {
    s = sl->Index(op_index_array[i])->GetSelection();
    ol->SetCur((EllipseSelection*) s);
    if (!ol->AtEnd()) {
        ts = ol->GetCur()->GetSelection()->GetTextSelection();
        if (ts != nil) {
            if (!sl->Find(ts)) {
                sl->Append(new SelectionNode(ts));
            }
        }
        met_ts = ol->GetCur()->GetSelection()->GetMETSelection();
        if (met_ts != nil) {
            if (!sl->Find(met_ts)) {
                sl->Append(new SelectionNode(met_ts));
            }
        }
    }
}

// For all operators selected, add their associated self loops and thei
// labels to the selection list

void Drawing::AddSelfLoopsToSelectionList() {
    int op_index = 0, sl_index = 0;
    int op_index_array[sl->Size()];
    for (sl->First(); !sl->AtEnd(); sl->Next()) {
        if (sl->GetCur()->GetSelection()->GetClassId() == OPERATOR) {
            op_index_array[op_index++] = sl_index;
        }
        ++sl_index;
    }

    Selection* s;
    DFDSplineSelList* dssl;
    for (int i = 0; i < op_index; ++i) {
        s = sl->Index(op_index_array[i])->GetSelection();
        ol->SetCur((EllipseSelection*) s);
        if (!ol->AtEnd()) {
            dssl = ol->GetCur()->GetSelection()->GetSelfLoopList();
            for (dssl->First(); !dssl->AtEnd(); dssl->Next()) {
                if (!sl->Find(dssl->GetCur()->GetSelection()->
                    GetSplineSelection())) {
                    sl->Append(new SelectionNode(dssl->GetCur()->
                        GetSelection()->GetSplineSelection()));
                }
            }
            TextSelection* ts =
                dssl->GetCur()->GetSelection()->GetTextSelection();

```

```

        if (ts != nil && !sl->Find(ts)) {
            printf("appending self lop's label to sl\n");
            if (ts->GetClassId() == LABEL_SL) {
                printf("its a self loop label\n");
            }
            else {
                if (ts->GetClassId() == LABEL_DF) {
                    printf("its a df loop\n");
                }
                else {
                    printf("we do not know what it is\n");
                }
            }
            sl->Append(new SelectionNode(ts));
        }
    }
}

// Modify all data flows associated with the given operator to move
// to the new location that the operator moved to

void Drawing::ReplaceAssociatedObjects(EllipseSelection* op, Drawing-
View* dv) {
    float fx, fy;
    Coord center_x, center_y;
    ol->SetCur(op);
    if (!ol->AtEnd()) {
        op->GetCenter(fx, fy);
        center_x = (Coord) fx;
        center_y = (Coord) fy;
/* ***** Start of Commented Out Code *****
        ReplaceInputDFLines(center_x, center_y, dv);
        ol->SetCur(op);
        ReplaceOutputDFLines(center_x, center_y, dv);
        ol->SetCur(op);
        ***** End of Commented Out Code ***** */
        ReplaceInputDFSplines(center_x, center_y, dv);
        ol->SetCur(op);
        ReplaceOutputDFSplines(center_x, center_y, dv);
    }
}

/* ***** Start of Commented Out Code *****

void Drawing::ReplaceInputDFLines(Coord cx, Coord cy, DrawingView* dv) {
    Coord x_new[2], y_new[2];
    Coord x_old_0, y_old_0, x_old_1, y_old_1, x_new_1, y_new_1;
    LineSelection* oldls;
    LineSelection* newls;
    DFDFLineSelList* ill =

```

```

        ol->GetCur()->GetSelection()->GetInputDFLineList();
int size = ill->Size();
int index = 0;
ill->First();
while (index < size) {
    oldls = ill->GetCur()->GetSelection()->GetLineSelection();
    TextSelection* ts =
        ill->GetCur()->GetSelection()->GetTextSelection();
    oldls->GetOriginal2(x_old_0, y_old_0, x_old_1, y_old_1);
    x_new_1 = cx;
    y_new_1 = cy;
    FindOperatorIntersection(x_new_1, y_new_1, x_old_0, y_old_0);
    x_new[0] = x_old_0;
    y_new[0] = y_old_0;
    x_new[1] = x_new_1;
    y_new[1] = y_new_1;
    newls = (LineSelection*) oldls
        ->CreateReshapedCopy(x_new, y_new, 2);
    ReplaceChange* rc = new ReplaceChange(this, dv, oldls, newls);
    rc->Do();
    ReplaceLabel(newls, ts);
//    SelectionList* temp = GetSelections();
//    SelectionList* align_sels = new SelectionList;
//    align_sels->Append(new SelectionNode(newls));
//    align_sels->Append(new SelectionNode(ts));
//    Select(align_sels);
//    Align(Center, Center);
//    Select(temp);
    ++index;
    ill->Index(index);
}
}

void Drawing::ReplaceOutputDFLines(Coord cx, Coord cy, DrawingView* dv) {
    Coord x_new[2], y_new[2];
    Coord x_old_0, y_old_0, x_old_1, y_old_1, x_new_0, y_new_0;
    LineSelection* oldls;
    LineSelection* newls;

    DFDLineSelList* oll =
        ol->GetCur()->GetSelection()->GetOutputDFLineList();
int size = oll->Size();
int index = 0;
oll->First();
while (index < size) {
    oldls = oll->GetCur()->GetSelection()->GetLineSelection();
    TextSelection* ts =
        oll->GetCur()->GetSelection()->GetTextSelection();
    oldls->GetOriginal2(x_old_0, y_old_0, x_old_1, y_old_1);
    x_new_0 = cx;
    y_new_0 = cy;
    FindOperatorIntersection(x_new_0, y_new_0, x_old_1, y_old_1);

```

```

        x_new[0] = x_new_0;
        y_new[0] = y_new_0;
        x_new[1] = x_old_1;
        y_new[1] = y_old_1;
        newls = (LineSelection*) oldls
                ->CreateReshapedCopy(x_new, y_new, 2);
        ReplaceChange* rc = new ReplaceChange(this, dv, oldls, newls);
        rc->Do();
        ReplaceLabel(newls, ts);
        ++index;
        oll->Index(index);
    }
}

***** End of Commented Out Code ***** */

// Modify all input data flows associated with an operator

void Drawing::ReplaceInputDFSplines(Coord cx, Coord cy, DrawingView* dv)
{
    Coord* x_new;
    Coord* y_new;
    Coord x_new_n, y_new_n;
    Coord* x_old;
    Coord* y_old;
    BSplineSelection* oldss;
    BSplineSelection* newss;

    DFDSplineSelList* isl =
        ol->GetCur()->GetSelection()->GetInputDFSplineList();
    int size = isl->Size();
    int index = 0;
    isl->First();
    while (index < size) {
        oldss = isl->GetCur()->GetSelection()->GetSplineSelection();
        TextSelection* ts =
            isl->GetCur()->GetSelection()->GetTextSelection();
        TextSelection* lat_ts =
            isl->GetCur()->GetSelection()->GetLatencySelection();
        int num = oldss->GetOriginal(x_old, y_old);
        x_new_n = cx;
        y_new_n = cy;
        FindOperatorIntersection(x_new_n, y_new_n, x_old[num-2],
                                y_old[num-2]);

        x_new = x_old;
        y_new = y_old;
        x_new[num-1] = x_new_n;
        y_new[num-1] = y_new_n;
        newss = (BSplineSelection*) oldss
                ->CreateReshapedCopy(x_new, y_new, num,
                                      DATAFLOW_SPLINE);
        ReplaceChange* rc = new ReplaceChange(this, dv, oldss, newss);
        rc->Do();
    }
}

```

```

        ReplaceLabel(newss, ts);
        if (ts != nil) {
            ReplaceLatency(ts, lat_ts);
        }
        else {
            ReplaceLatency(newss, lat_ts);
        }
        ++index;
        isl->Index(index);
    }
}

// Replace all output data flows connected to operator

void Drawing::ReplaceOutputDFSplines(Coord cx, Coord cy, DrawingView* dv)
{
    Coord* x_new;
    Coord* y_new;
    Coord x_new_0, y_new_0;
    Coord* x_old;
    Coord* y_old;
    BSplineSelection* oldss;
    BSplineSelection* newss;

    DFDSplineSelList* osl =
        ol->GetCur()->GetSelection()->GetOutputDFSplineList();
    int size = osl->Size();
    int index = 0;
    osl->First();
    while (index < size) {
        oldss = osl->GetCur()->GetSelection()->GetSplineSelection();
        TextSelection* ts =
            osl->GetCur()->GetSelection()->GetTextSelection();
        TextSelection* lat_ts =
            osl->GetCur()->GetSelection()->GetLatencySelection();
        int num = oldss->GetOriginal(x_old, y_old);
        x_new_0 = cx;
        y_new_0 = cy;
        FindOperatorIntersection(x_new_0, y_new_0, x_old[1], y_old[1]);
        x_new = x_old;
        y_new = y_old;
        x_new[0] = x_new_0;
        y_new[0] = y_new_0;
        newss = (BSplineSelection*) oldss
            ->CreateReshapedCopy(x_new, y_new, num
                                , DATAFLOW_SPLINE);
        ReplaceChange* rc = new ReplaceChange(this, dv, oldss, newss);
        rc->Do();
        ReplaceLabel(newss, ts);
        if (ts != nil) {
            ReplaceLatency(ts, lat_ts);
        }
    }
}

```

```

        else {
            ReplaceLatency(newss, lat_ts);
        }
        ++index;
        osl->Index(index);
    }
}

// Not called anymore, because self loops can be translated

void Drawing::ReplaceSelfLoops(Coord cx, Coord cy, DrawingView* dv) {
    Coord* x_new;
    Coord* y_new;
    Coord x_new_n, y_new_n, x_new_0, y_new_0;
    Coord* x_old;
    Coord* y_old;
    BSplineSelection* oldss;
    BSplineSelection* newss;

    DFDSplineSelList* sll =
        ol->GetCur()->GetSelection()->GetSelfLoopList();
    int size = sll->Size();
    int index = 0;
    sll->First();
    while (index < size) {
        oldss = sll->GetCur()->GetSelection()->GetSplineSelection();
        int num = oldss->GetOriginal(x_old, y_old);
        x_new_n = cx;
        y_new_n = cy;
        x_new_0 = cx;
        y_new_0 = cy;
        FindOperatorIntersection(x_new_n, y_new_n, x_old[num-2],
                                y_old[num-2]);
        FindOperatorIntersection(x_new_0, y_new_0, x_old[1], y_old[1]);
        x_new = x_old;
        y_new = y_old;
        x_new[num-1] = x_new_n;
        y_new[num-1] = y_new_n;
        x_new[0] = x_new_0;
        y_new[0] = y_new_0;
        newss = (BSplineSelection*) oldss
            ->CreateReshapedCopy(x_new, y_new, num,
                                SELF_LOOP);

        ReplaceChange* rc = new ReplaceChange(this, dv, oldss, newss);
        rc->Do();
        ++index;
        sll->Index(index);
    }
}

// Write the PSDL specification of the given operator to a temporary
// file to be edited

```



```

void Drawing::WritePSDLForOperator(EllipseSelection* op) {
    ol->SetCur(op);
    if (!ol->AtEnd()) {
        char* filename = MakeTmpFileName(PSDL_FILE);
        FILE* fptr = fopen(filename, "w");
        char* psdl = ol->GetCur()->GetSelection()->GetPSDLText();
        fprintf(fptr, "%s", psdl);
        fclose(fptr);
        delete filename;
        delete psdl;
    }
}

void Drawing::ReplaceLabel(Selection* newsel, TextSelection* ts) {
    if (ts != nil) {
        SelectionList* temp = GetSelections();
        SelectionList* align_sels = new SelectionList;
        align_sels->Append(new SelectionNode(newsel));
        align_sels->Append(new SelectionNode(ts));
        Select(align_sels);
        Align(Center, Center);
        Select(temp);
    }
}

void Drawing::ReplaceLatency(Selection* newsel, TextSelection* lat_ts) {
    if (lat_ts != nil) {
        SelectionList* temp = GetSelections();
        SelectionList* align_sels = new SelectionList;
        align_sels->Append(new SelectionNode(newsel));
        align_sels->Append(new SelectionNode(lat_ts));
        Select(align_sels);
        Align(Center, Center);
        Align(Top, Bottom);
        Select(temp);
    }
}

// add met to operator selection list

void Drawing::METAppend(TextSelection* met_ts, EllipseSelection* op) {
    ol->AddMETToOperator(met_ts, op);
}

// add latency to data flow in operator list

void Drawing::LatencyAppend(TextSelection* lat_ts, BSplineSelection* df)
{
    ol->AddLatencyToDataFlow(lat_ts, df);
}

// change PSDL specification of drawing to use given prototype name as

```

```

// operator id

void Drawing::UpdatePSDLSpec(const char* new_prototype_name) {
    int index = spec_txb->ForwardSearch(new Regexp(OPER_TKN), 0);
    if (index >= 0) {
        int end_index = spec_txb->EndOfLine(index);
        spec_txb->Delete(index, end_index - index);
        if (new_prototype_name == nil) {
            spec_txb->Insert(index, ID_TKN, strlen(ID_TKN));
        }
        else {
            spec_txb->Insert(index, new_prototype_name,
                             strlen(new_prototype_name));
        }
    }
}

// write PSDL specification of drawing to temporary file to be edited

void Drawing::WritePSDLForDrawing() {
    const char* text = spec_txb->Text();
    char* filename = MakeTmpFileName(PSDL_FILE);
    FILE* psdl = fopen(filename, "w");
    fprintf(psdl, "%s", text);
    fclose(psdl);
    delete filename;
}

// write the PSDL graph to a file

EdgeList* Drawing::WritePSDLGraph(FILE* iptr) {
    fprintf(iptr, IMP_TKN);
    fprintf(iptr, GR_TKN);
    WriteVertices(iptr);
    EdgeList* el = WriteEdges(iptr);
    return el;
}

// write PSDL edges to file containing PSDL graph

EdgeList* Drawing::WriteEdges(FILE* fptr) {
    EdgeList* el = BuildEdgeList();

    for (el->First(); !el->AtEnd(); el->Next()) {
        char* froml = el->GetCur()->GetSelection()->GetFromVertexLabel();
        char* toL = el->GetCur()->GetSelection()->GetToVertexLabel();
        char* label = el->GetCur()->GetSelection()->GetEdgeLabel();
        char* f_fixed_string = RemoveBadChars(froml);
        char* t_fixed_string = RemoveBadChars(toL);
        char* latency = el->GetCur()->GetSelection()
                        ->GetEdgeLatency();
        char* fixed_label = RemoveBadChars(label);
    }
}

```

```

        if (latency != nil) {
            fprintf(fp_ptr, "%s %s : %s %s -> %s\n", EDGE_TKN, fixed_label,
                    latency, f_fixed_string, t_fixed_string);
        }
        else {
            fprintf(fp_ptr, "%s %s %s -> %s\n", EDGE_TKN, fixed_label,
                    f_fixed_string, t_fixed_string);
        }
        delete f_fixed_string;
        delete t_fixed_string;
        delete fixed_label;
    }
    return el;
}

// write PSDL vertices to file containing PSDL graph

void Drawing::WriteVertices(FILE* fp_ptr) {
    for (ol->First(); !ol->AtEnd(); ol->Next()) {
        TextSelection* ts = ol->GetCur()->GetSelection()->GetTextSelection();
        char* label;
        const char* tmp;
        char* met;
        int len;
        if (ts != nil) {
            tmp = ts->GetOriginal(len);
            label = new char[len+1];
            strncpy(label, tmp, len);
            label[len] = '\0';
        }
        else {
            label = ID_TKN;
        }
        printf("in write vertices, label is %s\n", label);

        boolean hasMET = false;
        ts = ol->GetCur()->GetSelection()->GetMETSelection();
        if (ts != nil) {
            hasMET = true;
            tmp = ts->GetOriginal(len);
            met = new char[len+1];
            strncpy(met, tmp, len);
            met[len] = '\0';
        }
        char* fixed_string = RemoveBadChars(label);
        fprintf(fp_ptr, "%s %s ", VER_TKN, fixed_string);
        delete fixed_string;
        if (hasMET)
            fprintf(fp_ptr, ": %s\n", met);
        else

```

```

        fprintf(fptr, "\n");
    }
}

// construct list of edges with associated input and output operators
// to be used when creating PSDL graph

EdgeList* Drawing::BuildEdgeList() {
    EdgeList* el = new EdgeList;
    for (ol->First(); !ol->AtEnd(); ol->Next()) {
        OperatorSelection* op = ol->GetCur()->GetSelection();

        DFDSplineSelList* insl =
            ol->GetCur()->GetSelection()->GetInputDFSplineList();
        for (insl->First(); !insl->AtEnd(); insl->Next()) {
            if (!el->FindSpline(insl->GetCur()->GetSelection())) {
                Edge* edge1 = new Edge(insl->GetCur()->GetSelection());
                edge1->SetToVertex(op);
                el->Append(new EdgeNode(edge1));
            }
            else {
                el->GetCur()->GetSelection()->SetToVertex(op);
            }
        }

        DFDSplineSelList* outsl =
            ol->GetCur()->GetSelection()->GetOutputDFSplineList();
        for (outsl->First(); !outsl->AtEnd(); outsl->Next()) {
            if (!el->FindSpline(outsl->GetCur()->GetSelection())) {
                Edge* edge2 = new Edge(outsl->GetCur()->GetSelection());
                edge2->SetFromVertex(op);
                el->Append(new EdgeNode(edge2));
            }
            else {
                el->GetCur()->GetSelection()->SetFromVertex(op);
            }
        }
    }
    return el;
}

// write psdl specification file and psdl implementation file and write
// psdl specification file for all the operators and write file needed
// to rebuild operator list when reentering editor

void Drawing::WriteDFDFiles(char* dir, const char* prototype_name) {
    char* spec_filename = new char[strlen(dir) + strlen(prototype_name)
                                    + SPEC_EXT_LEN + 1];

    strcpy(spec_filename, dir);
    strcat(spec_filename, prototype_name);
    strcat(spec_filename, SPEC_PSDL_EXT);
    FILE* sptr = fopen(spec_filename, "w");
}

```

```

const char* spec_string = spec_txb->Text();
fprintf(sptr, "%s", spec_string);
fclose(sptr);
delete spec_filename;

char* imp_filename = new char[strlen(dir) + strlen(prototype_name)
                                + IMP_EXT_LEN + 1];
strcpy(imp_filename, dir);
strcat(imp_filename, prototype_name);
strcat(imp_filename, IMP_PSDL_EXT);
FILE* iptr = fopen(imp_filename, "w");

EdgeList* el = WritePSDLGraph(iptr);
if (streams_txb != nil) {
    const char* streams_string = streams_txb->Text();
    fprintf(iptr, "%s", streams_string);
}
if (constraints_txb != nil) {
    const char* con_string = constraints_txb->Text();
    fprintf(iptr, "%s", con_string);
}
fprintf(iptr, "%s", END_TKN);
fclose(iptr);
delete imp_filename;

WritePSDLForAllOperators(dir, prototype_name);

char* graph_filename = new char [strlen(dir) + strlen(prototype_name) +
                                DFD_EXT_LEN + 1];
strcpy(graph_filename, dir);
strcat(graph_filename, prototype_name);
strcat(graph_filename, DFD_EXT);
FILE* gptr = fopen(graph_filename, "w");
WriteDFDInfo(gptr, el);
fclose(gptr);
delete graph_filename;
}
// read psdl specification file and psdl implementation file and read
// psdl specification file for all the operators and read in file to
// rebuild operator list

void Drawing::ReadDFDFiles(char* dir, const char* prototype_name) {
    printf("in ReadDFDFiles\n");
    char* spec_filename = new char[strlen(dir) + strlen(prototype_name)
                                    + SPEC_EXT_LEN + 1];
    strcpy(spec_filename, dir);
    strcat(spec_filename, prototype_name);
    strcat(spec_filename, SPEC_PSDL_EXT);
    printf("spec_filename is %s\n", spec_filename);
    FILE* sptr = fopen(spec_filename, "r");
    if (sptr != nil) {

```

```

        char* spec_string = ReadBackPSDL(sptr);
        fclose(sptr);
        if (spec_string != nil) {
            printf("spec_string is %s\n", spec_string);
            if (spec_txb != nil)
                delete spec_txb;
            spec_txb = new TextBuffer(spec_string, strlen(spec_string),
                                      TXTBUFLen);
        }
    }
    delete spec_filename;

    char* imp_filename = new char[strlen(dir) + strlen(prototype_name)
                                   + IMP_EXT_LEN + 1];
    strcpy(imp_filename, dir);
    strcat(imp_filename, prototype_name);
    strcat(imp_filename, IMP_PSDL_EXT);
    printf("imp_filename is %s\n", imp_filename);
    FILE* iptr = fopen(imp_filename, "r");
    if (iptr != nil) {
        char* imp_string = ReadBackPSDL(iptr);
        fclose(iptr);
        if (imp_string != nil) {
            printf("imp_string is %s\n", imp_string);
            FillImpBuffers(imp_string);
        }
    }
    delete imp_filename;

    char* graph_filename = new char [strlen(dir) + strlen(prototype_name) +
                                     DFD_EXT_LEN + 1];
    strcpy(graph_filename, dir);
    strcat(graph_filename, prototype_name);
    strcat(graph_filename, DFD_EXT);
    printf("graph_filename is %s\n", graph_filename);
    FILE* gptr = fopen(graph_filename, "r");
    if (gptr != nil) {
        ReadDFDInfo(gptr);
        fclose(gptr);
    }
    delete graph_filename;

    ReadPSDLForAllOperators(dir, prototype_name);
}

// write PSDL specificatio for each operator to a file created by conte-
// nating
// the operator's identifier with the prototype name

void Drawing::WritePSDLForAllOperators(char* dir, const char* prototype_
name) {

```

```

        for (ol->First(); !ol->AtEnd(); ol->Next()) {
            TextSelection* ts = ol->GetCur()->GetSelection()->GetTextSelection();
            if (ts != nil) {
                int len;
                const char* tmp_string = ts->GetOriginal(len);
                char* op_name = new char[len + 1];
                strncpy(op_name, tmp_string, len);
                op_name[len] = '\0';
                char* fixed_name = RemoveBadChars(op_name);
                char* filename = new char[strlen(dir) + strlen(prototype_name)
                    + strlen(fixed_name)
                    + SPEC_EXT_LEN + 2];

                strcpy(filename, dir);
                strcat(filename, prototype_name);
                strcat(filename, ".");
                strcat(filename, fixed_name);
                strcat(filename, SPEC_PSDL_EXT);
                FILE* fptr = fopen(filename, "w");
                char* text = ol->GetCur()->GetSelection()->GetPSDLText();
                fprintf(fptr, "%s", text);
                fclose(fptr);
                delete op_name;
                delete fixed_name;
                delete filename;
                delete text;
            }
        }
    }

    // write contents of streams buffer to file

    void Drawing::WriteStreams() {
        char* filename = MakeTmpFileName(STREAMS_FILE);
        FILE* fptr = fopen(filename, "w");
        if (streams_txb != nil) {
            const char* text = streams_txb->Text();
            fprintf(fptr, "%s", text);
        }
        fclose(fptr);
        delete filename;
    }

    // write contents of constraints buffer to file

    void Drawing::WriteConstraints() {
        char* filename = MakeTmpFileName(CONSTRAINTS_FILE);
        FILE* fptr = fopen(filename, "w");
        if (constraints_txb != nil) {
            const char* text = constraints_txb->Text();
            fprintf(fptr, "%s", text);
        }
    }

```

```

        fclose(fptr);
        delete filename;
    }

    // after user edits psdl for operator, must read back into operator's
    // text buffer

    void Drawing::ReadPSDLForOperator(FILE* fptr, EllipseSelection* op) {
        char* text = ReadBackPSDL(fptr);
        if (text != nil) {
            ol->SetCur(op);
            if (!ol->AtEnd()) {
                ol->GetCur()->GetSelection()->SetPSDLText(text);
            }
        }
    }

    // after user edits psdl for drawing, must read back into specification
    // text buffer

    void Drawing::ReadPSDLForDrawing() {
        char* filename = MakeTmpFileName(PSDL_FILE);
        FILE* fptr = fopen(filename, "r");
        if (fptr != nil) {
            char* text = ReadBackPSDL(fptr);
            fclose(fptr);
            if (text != nil) {
                if (spec_txb != nil)
                    delete spec_txb;
                spec_txb = new TextBuffer(text, strlen(text), TXTBUFLLEN);
            }
            delete filename;
        }
    }

    // read file containing PSDL

    char* Drawing::ReadBackPSDL(FILE* fptr) {
        boolean first_time = true;
        char* text = nil;
        char* line = new char [200];
        while (fgets(line, 200, fptr) != nil) {
            if (first_time) {
                text = new char[TXTBUFLLEN];
                strcpy(text, line);
                first_time = false;
            }
            else {
                strcat(text, line);
            }
        }
        delete line;
    }

```



```

        return text;
    }

    // after user edits psdl streams, must read back into streams text buffer

void Drawing::ReadStreams() {
    char* filename = MakeTmpFileName(STREAMS_FILE);
    FILE* fptr = fopen(filename, "r");
    if (fptr != nil) {
        char* text = ReadBackPSDL(fptr);
        fclose(fptr);
        if (text != nil) {
            if (streams_txb != nil) {
                delete streams_txb;
            }
            streams_txb = new TextBuffer(text, strlen(text), TXTBUFLLEN);
        }
    }
    delete filename;
}

    // after user edits psdl constraints, must read back into constraints text
    // buffer

void Drawing::ReadConstraints() {
    char* filename = MakeTmpFileName(CONSTRAINTS_FILE);
    FILE* fptr = fopen(filename, "r");
    if (fptr != nil) {
        char* text = ReadBackPSDL(fptr);
        fclose(fptr);
        if (text != nil) {
            if (constraints_txb != nil) {
                delete constraints_txb;
            }
            constraints_txb = new TextBuffer(text, strlen(text), TXTBUFLLEN);
        }
    }
    delete filename;
}

    // write information to file to be able to rebuild operator list when
    // coming back into editor

void Drawing::WriteDFDInfo(FILE* fptr, EdgeList* el) {
    TextSelection* ts;
    for (ol->First(); !ol->AtEnd(); ol->Next()) {
        OperatorSelection* op = ol->GetCur()->GetSelection();
        int pict_op_index = picture->FindIndex(op->GetEllipseSelection());
        WriteOperator(fptr, pict_op_index);

        ts = op->GetTextSelection();
        if (ts != nil) {

```

```

        int pict_label_index = picture->FindIndex(ts);
        WriteOpLabel(fptr, pict_op_index, pict_label_index);
    }

    ts = op->GetMETSelection();
    if (ts != nil) {
        int pict_MET_index = picture->FindIndex(ts);
        WriteMET(fptr, pict_op_index, pict_MET_index);
    }

    DFDSplineSelList* sll = op->GetSelfLoopList();
    for (sll->First(); !sll->AtEnd(); sll->Next()) {
        BSplineSelection* selfloop = sll->GetCur()->GetSelection()
                                     ->GetSplineSelection();

        int pict_sl_index = picture->FindIndex(selfloop);
        WriteSelfLoop(fptr, pict_op_index, pict_sl_index);

        ts = sll->GetCur()->GetSelection()->GetTextSelection();
        if (ts != nil) {
            int pict_sl_label_index = picture->FindIndex(ts);
            WriteSelfLoopLabel(fptr, pict_op_index, pict_sl_index,
                               pict_sl_label_index);
        }
    }
}

for (el->First(); !el->AtEnd(); el->Next()) {
    boolean is_stream;
    BSplineSelection* flow = el->GetCur()->GetSelection()
                           ->GetEdge()->GetSplineSelection();
    is_stream = flow->IsAStream();
    int pict_flow_index = picture->FindIndex(flow);
    OperatorSelection* input_op = el->GetCur()->GetSelection()
                                  ->GetToVertex();

    int pict_flow_iop_index;
    if (input_op != nil) {
        pict_flow_iop_index =
            picture->FindIndex(input_op->GetEllipseSelection());
    }
    else {
        pict_flow_iop_index = -1;
    }
    OperatorSelection* output_op = el->GetCur()->GetSelection()
                                   ->GetFromVertex();

    int pict_flow_oop_index;
    if (output_op != nil) {
        pict_flow_oop_index =
            picture->FindIndex(output_op->GetEllipseSelection());
    }
    else {
        pict_flow_oop_index = -1;
    }
}

```

```

        WriteFlow(fp_ptr, pict_flow_index, pict_flow_iop_index,
                  pict_flow_oop_index, is_stream);

        ts = el->GetCur()->GetSelection()->GetEdge()->GetTextSelection();
        if (ts != nil) {
            int pict_flow_label_index = picture->FindIndex(ts);
            WriteFlowLabel(fp_ptr, pict_flow_index, pict_flow_iop_index,
                           pict_flow_oop_index, pict_flow_label_index);
        }
        ts = el->GetCur()->GetSelection()->GetEdge()->GetLatencySelection();
        if (ts != nil) {
            int pict_flow_lat_index = picture->FindIndex(ts);
            WriteFlowLatency(fp_ptr, pict_flow_index, pict_flow_iop_index,
                              pict_flow_oop_index, pict_flow_lat_index);
        }
    }
}

// write operator information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteOperator(FILE* fp_ptr, int op_index) {
    fprintf(fp_ptr, "%u\n", OPERATOR);
    fprintf(fp_ptr, "%d\n", op_index);
}

// write operator label information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteOpLabel(FILE* fp_ptr, int op_index, int label_index) {
    fprintf(fp_ptr, "%u\n", LABEL_OP);
    fprintf(fp_ptr, "%d %d\n", op_index, label_index);
}

// write operator met information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteMET(FILE* fp_ptr, int op_index, int MET_index) {
    fprintf(fp_ptr, "%u\n", MET_OP);
    fprintf(fp_ptr, "%d %d\n", op_index, MET_index);
}

// write selfloop information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteSelfLoop(FILE* fp_ptr, int op_index, int sl_index) {
    fprintf(fp_ptr, "%u\n", SELFLOOP);
    fprintf(fp_ptr, "%d %d\n", op_index, sl_index);
}

// write selfloop label information to file containing DFD information to

```

```

// be able to rebuild operator list

void Drawing::WriteSelfLoopLabel(FILE* fptr, int op_index, int sl_index,
                                int sl_label_index) {
    fprintf(fptr, "%u\n", LABEL_SL);
    fprintf(fptr, "%d %d %d\n", op_index, sl_index, sl_label_index);
}

// write data flow information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteFlow(FILE* fptr, int flow_index, int input_op_index,
                        int output_op_index, boolean is_stream) {
    fprintf(fptr, "%u\n", DATAFLOW_SPLINE);
    fprintf(fptr, "%d %d %d %u\n", flow_index, input_op_index,
        output_op_index, is_stream);
}

// write data flow label information to file containing DFD information to
// be able to rebuild operator list

void Drawing::WriteFlowLabel(FILE* fptr, int flow_index, int input_op_index,
                             int output_op_index, int flow_label_index) {
    fprintf(fptr, "%u\n", LABEL_DF);
    fprintf(fptr, "%d %d %d %d\n", flow_index, input_op_index,
        output_op_index, flow_label_index);
}

// write data flow latency information to file containing DFD information
// to
// be able to rebuild operator list

void Drawing::WriteFlowLatency(FILE* fptr, int flow_index, int input_op_index,
                               int output_op_index, int flow_lat_index) {
    fprintf(fptr, "%u\n", LAT_DF);
    fprintf(fptr, "%d %d %d %d\n", flow_index, input_op_index,
        output_op_index, flow_lat_index);
}

// Read in file to rebuild operator list

void Drawing::ReadDFDInfo(FILE* fptr) {
    ClassId cid;
    while (fscanf(fptr, "%u", &cid) != EOF) {

        // append the proper element to the operator list based on its class id
        // read

        EllipseSelection* op;
        EllipseSelection* input_op;
        EllipseSelection* output_op;
    }
}

```

```

BSplineSelection* sl;
BSplineSelection* df;
TextSelection* ts;
int op_index, label_index, flow_index, sl_index, MET_index;
int input_op_index, output_op_index;
switch (cid) {
case OPERATOR:
    fscanf(fp_ptr, "%d", &op_index);
    op = (EllipseSelection*) picture->GetSelection(op_index);
    op->SetClassId(OPERATOR);
    OperatorAppend(op);
    break;
case LABEL_OP:
    fscanf(fp_ptr, "%d %d", &op_index, &label_index);
    op = (EllipseSelection*) picture->GetSelection(op_index);
    ts = (TextSelection*) picture->GetSelection(label_index);
    ts->SetClassId(LABEL_OP);
    LabelReadAppend(ts, op);
    break;
case MET_OP:
    fscanf(fp_ptr, "%d %d", &op_index, &MET_index);
    op = (EllipseSelection*) picture->GetSelection(op_index);
    ts = (TextSelection*) picture->GetSelection(MET_index);
    ts->SetClassId(MET_OP);
    METAppend(ts, op);
    break;
case SELFLOOP:
    fscanf(fp_ptr, "%d %d", &op_index, &sl_index);
    op = (EllipseSelection*) picture->GetSelection(op_index);
    sl = (BSplineSelection*) picture->GetSelection(sl_index);
    sl->SetClassId(SELFLOOP);
    DataFlowSplineAppend(sl, op, op);
    break;
case LABEL_SL:
    fscanf(fp_ptr, "%d %d %d", &op_index, &sl_index, &label_index);
    sl = (BSplineSelection*) picture->GetSelection(sl_index);
    ts = (TextSelection*) picture->GetSelection(label_index);
    ts->SetClassId(LABEL_SL);
    LabelReadAppend(ts, sl);
    break;
case DATAFLOW_SPLINE:
    boolean is_stream;
    fscanf(fp_ptr, "%d %d %d %u", &flow_index, &input_op_index,
                                   &output_op_index, &is_stream);

    input_op = nil;
    if (input_op_index != -1) {
        input_op =
            (EllipseSelection*) picture->GetSelection(input_op_in-
dex);
    }
    output_op = nil;
    if (output_op_index != -1) {

```

```

        output_op =
            (EllipseSelection*) picture->GetSelection(output_op_in-
dex);
    }
    df = (BSplineSelection*) picture->GetSelection(flow_index);
    df->SetClassId(DATAFLOW_SPLINE);
    if (is_stream)
        df->SetStream();
    DataFlowSplineAppend(df, output_op, input_op);
    break;
case LABEL_DF:
    fscanf(fptr, "%d %d %d %d", &flow_index, &input_op_index,
        &output_op_index, &label_index);
    df = (BSplineSelection*) picture->GetSelection(flow_index);
    ts = (TextSelection*) picture->GetSelection(label_index);
    ts->SetClassId(LABEL_DF);
    LabelReadAppend(ts, df);
    break;
case LAT_DF:
    int lat_index;
    fscanf(fptr, "%d %d %d %d", &flow_index, &input_op_index,
        &output_op_index, &lat_index);
    df = (BSplineSelection*) picture->GetSelection(flow_index);
    ts = (TextSelection*) picture->GetSelection(lat_index);
    ts->SetClassId(LAT_DF);
    LatencyAppend(ts, df);
    break;
}
}

// if an element does not have a class id now, assume it is a comment
for (picture->First(); !picture->AtEnd(); picture->Next()) {
    if (picture->GetCurrent()->GetClassId() == NONE) {
        picture->GetCurrent()->SetClassId(COMMENT);
    }
}

// Read PSDL back into all operators' text buffers

void Drawing::ReadPSDLForAllOperators(char* dir, const char* prototype_
name) {
    for (ol->First(); !ol->AtEnd(); ol->Next()) {
        TextSelection* ts = ol->GetCur()->GetSelection()->GetTextSele-
tion();
        if (ts != nil) {
            int len;
            const char* tmp_string = ts->GetOriginal(len);
            char* op_name = new char[len + 1];
            strncpy(op_name, tmp_string, len);
            op_name[len] = '\0';

```

```

        char* fixed_name = RemoveBadChars(op_name);
        char* filename = new char[strlen(dir) + strlen(prototype_name)
                                   + strlen(fixed_name)
                                   + SPEC_EXT_LEN + 2];

        strcpy(filename, dir);
        strcat(filename, prototype_name);
        strcat(filename, ".");
        strcat(filename, fixed_name);
        strcat(filename, SPEC_PSDL_EXT);
        FILE* fptr = fopen(filename, "r");
        if (fptr != nil) {
            char* text = ReadBackPSDL(fptr);
            fclose(fptr);
            ol->GetCur()->GetSelection()->SetPSDLText(text);
        }
        delete op_name;
        delete fixed_name;
        delete filename;
    }
}

// Return label of selected operator

char* Drawing::OperatorLabelIs(EllipseSelection* op) {
    ol->SetCur(op);
    if (!ol->AtEnd()) {
        TextSelection* ts = ol->GetCur()->GetSelection()->GetTextSelection();
        if (ts != nil) {
            int len;
            const char* temp_string = ts->GetOriginal(len);
            char* label = new char[len + 1];
            strncpy(label, temp_string, len);
            label[len] = '\0';
            char* fixed_string = RemoveBadChars(label);
            delete label;
            return fixed_string;
        }
    }
    return nil;
}

// Add type declaration to PSDL streams buffer

void Drawing::AddStream() {
    char* streams;
    if (streams_txb == nil) {
        streams = new char[TEXTBUFLLEN];
        strcpy(streams, STREAM_TKN);
        strcat(streams, TYPE_DECL_TKN);
        strcat(streams, "\n");
    }
}

```

```

        streams_txb = new TextBuffer(streams, strlen(streams), TXTBUFLLEN);
    }
    else {
        int index = streams_txb->ForwardSearch(new Regexp(STREAM_SCH_TKN),
                                                    0);

        if (index >= 0) {
            streams = new char[strlen(TYPE_DECL_TKN) + 3];
            strcpy(streams, TYPE_DECL_TKN);
            strcat(streams, ",\n");
            streams_txb->Insert(index+1, streams, strlen(streams));
        }
    }
}

// fill the stream PSDL buffer and the constraints PSDL buffer with
// the part of the given string that relates to them

void Drawing::FillImpBuffers(char* imp_string) {
    TextBuffer* imp_txb = new TextBuffer(imp_string, strlen(imp_string),
                                          TXTBUFLLEN);

    char* start_array_1[] = {STREAM_SCH_TKN, TIMER_SCH_TKN};
    int start_size = 2, start_index = -1;
    char* end_array_1[] = {CON_SCH_TKN, DESC_SCH_TKN, END_SCH_TKN};
    int end_size = 3, end_index = -1;
    int line_start_index, line_end_index;
    FindTxbIndices(imp_txb, start_array_1, start_size, start_index,
                  end_array_1, end_size, end_index);
    if (start_index >= 0 && end_index >= 0 && end_index > start_index) {
        line_start_index = imp_txb->BeginningOfLine(start_index);
        line_end_index = imp_txb->EndOfPreviousLine(end_index);
        char* streams_string = new char[TXTBUFLLEN];
        strncpy(streams_string, (imp_string+line_start_index),
                line_end_index-line_start_index+1);
        streams_string[line_end_index-line_start_index+1] = '\0';
        if (streams_txb != nil) {
            delete streams_txb;
        }
        streams_txb = new TextBuffer(streams_string, strlen(-
streams_string),
                                    TXTBUFLLEN);
    }

    char* start_array_2[] = {CON_SCH_TKN, DESC_SCH_TKN};
    start_size = 2;
    start_index = -1;
    char* end_array_2[] = {END_SCH_TKN};
    end_size = 1;
    end_index = -1;
    FindTxbIndices(imp_txb, start_array_2, start_size, start_index,
                  end_array_2, end_size, end_index);
    if (start_index >= 0 && end_index >= 0) {
        line_start_index = imp_txb->BeginningOfLine(start_index);
        line_end_index = imp_txb->EndOfPreviousLine(end_index);
    }
}

```



```

        char* con_string = new char[TXTBUFLen];
        strncpy(con_string, (imp_string+line_start_index),
                line_end_index-line_start_index+1);
        con_string[line_end_index-line_start_index+1] = '\0';
        if (constraints_txb != nil) {
            delete constraints_txb;
        }
        constraints_txb = new TextBuffer(con_string, strlen(con_string),
                TXTBUFLen);
    }
    delete imp_txb;
}

// find the start index and end index for extracting a portion of the
// given text buffer based on the array of starting strings and the array
// of ending strings

void Drawing::FindTxbIndices(TextBuffer* txb, char** start_array,
                             int start_size, int& start_index,
                             char** end_array, int end_size,
                             int& end_index) {
    for (int i = 0; i < start_size && start_index < 0; ++i) {
        start_index = txb->Search(new Regexp(start_array[i]), 0,
                TXTBUFLen, TXTBUFLen);
    }
    if (start_index >= 0) {
        for (i = 0; i < end_size && end_index < 0; ++i) {
            end_index = txb->Search(new Regexp(end_array[i]), 0,
                    TXTBUFLen, TXTBUFLen);
        }
    }
}

void Drawing::AddLabelToStreams(char* old_name, TextSelection* new_ts) {
    char* new_name;
    if (new_ts != nil) {
        int len;
        const char* tmp = new_ts->GetOriginal(len);
        new_name = new char[len+1];
        strncpy(new_name, tmp, len);
        new_name[len] = '\0';
    }
    else {
        new_name = ID_TKN;
    }
    char* new_fixed_name = RemoveBadChars(new_name);
    char* old_fixed_name = RemoveBadChars(old_name);

    int end_index = streams_txb->Search(new Regexp(TIMER_SCH_TKN),
                                        0, TXTBUFLen, TXTBUFLen);
    if (end_index < 0) {
        end_index = TXTBUFLen;
    }
}

```

```

    }
    int index = streams_txb->Search(new Regexp(old_fixed_name), 0, end_in-
dex,
                                     end_index);

    if (index >= 0) {
        streams_txb->Delete(index, strlen(old_fixed_name));
        streams_txb->Insert(index, new_fixed_name, strlen(new_fixed_
name));
    }
}

Selection* Drawing::FindOperator(TextSelection* lab_ts) {
    ol->SetCurWithLabel(lab_ts);
    if (!ol->AtEnd()) {
        return ol->GetCur()->GetSelection()->GetEllipseSelection();
    }
    return lab_ts;
}

Selection* Drawing::FindLabel(Selection* sel) {
    return ol->FindLabel(sel);
}

// Remove stream type declaration from PSDL

void Drawing::RemoveStream(TextSelection* ts) {
    char* id;
    if (ts != nil) {
        int len;
        const char* tmp = ts->GetOriginal(len);
        id = new char[len + 1];
        strncpy(id, tmp, len);
        id[len] = '\\0';
    }
    else {
        id = ID_TKN;
    }
    char* fixed_id = RemoveBadChars(id);
    int start_index = streams_txb->Search(new Regexp(STREAM_SCH_TKN), 0,
TXTBUFLen, TXTBUFLen);

    if (start_index >= 0) {
        int end_index = streams_txb->Search(new Regexp(TIMER_SCH_TKN),
0, TXTBUFLen, TXTBUFLen);

        if (end_index < 0) {
            end_index = TXTBUFLen;
        }
        if (end_index > start_index) {
            int index = streams_txb->Search(new Regexp(fixed_id), start_in-
dex,
                                     end_index - start_index + 1,
                                     end_index);

```

```

        if (index >= 0) {
            int end_line_index = streams_txb->EndOfLine(index);
            int start_line_index = streams_txb->BeginningOfLine(index);
            streams_txb->Delete(start_line_index,
                               end_line_index - start_line_index + 1);
        }
    }
}

// remove all of the self loops and data flows associated with
// an operator

void Drawing::RemoveAssociatedObjects(SelectionList* sl_2,
                                      EllipseSelection* op) {
    SelectionList* sel_list = new SelectionList;
    ol->SetCur(op);
    DFDSplineSelList* sll = ol->GetCur()->GetSelection()
                               ->GetSelfLoopList();
    for (sll->First(); !sll->AtEnd(); sll->Next()) {
        if (!sl_2->Find(sll->GetCur()->GetSelection()->
                       GetSplineSelection())) {
            sel_list->Append(new SelectionNode(sll->GetCur()
                                               ->GetSelection()->GetSplineSelection()));
        }
    }
    DFDSplineSelList* isl = ol->GetCur()->GetSelection()
                               ->GetInputDFSplineList();
    for (isl->First(); !isl->AtEnd(); isl->Next()) {
        if (!sl_2->Find(isl->GetCur()->GetSelection()->
                       GetSplineSelection())) {
            sel_list->Append(new SelectionNode(isl->GetCur()
                                               ->GetSelection()->GetSplineSelection()));
        }
    }
    DFDSplineSelList* osl = ol->GetCur()->GetSelection()
                               ->GetOutputDFSplineList();
    for (osl->First(); !osl->AtEnd(); osl->Next()) {
        if (!sl_2->Find(osl->GetCur()->GetSelection()->
                       GetSplineSelection())) {
            sel_list->Append(new SelectionNode(osl->GetCur()
                                               ->GetSelection()->GetSplineSelection()));
        }
    }
    RemoveSplines(sel_list);
}

// Remove a selection list filled with the associated splines of an
// operator

void Drawing::RemoveSplines(SelectionList* sel_list) {
    for (sel_list->First(); !sel_list->AtEnd(); sel_list->Next()) {
        Selection* s = sel_list->GetCur()->GetSelection();
    }
}

```

```

TextSelection* ts1 = nil;
TextSelection* ts2 = nil;
ClassId cid = s->GetClassId();
switch (cid) {
case DATAFLOW_SPLINE:
    ts1 = ol->GetDFLabel((BSplineSelection*) s);
    ts2 = ol->GetDFLatency((BSplineSelection*) s);
    if (((BSplineSelection*) s) ->IsAStream()) {
        RemoveStream(ts1);
    }
    break;
case SELFLOOP:
    ts1 = ol->GetSelfLoopLabel((BSplineSelection*) s);
    break;
default:
    break;
}
ol->Remove(s);
if (ts1 != nil) {
    picture->Remove(ts1);
}
if (ts2 != nil) {
    picture->Remove(ts2);
}
picture->Remove(s);
}
}

```

```

// file      edge.h
// description: Class description of Edge class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * this header file was made specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last Change made:  October 1, 1990
 */

#ifndef edge_h
#define edge_h

class DFDSplineSelection;
class OperatorSelection;

/*
 * definition of class storing the actual data flow and the labels
 * associated with its input and output operators
 */

class Edge {
public:
    Edge(DFDSplineSelection*);
    void SetFromVertex(OperatorSelection* o) { fromv = o; }
    void SetToVertex(OperatorSelection* i) { tov = i; }
    OperatorSelection* GetFromVertex() { return fromv; }
    OperatorSelection* GetToVertex() { return tov; }
    char* GetEdgeLabel();
    char* GetEdgeLatency();
    DFDSplineSelection* GetEdge() { return flow; }
    char* GetFromVertexLabel();
    char* GetToVertexLabel();

protected:
    DFDSplineSelection* flow;
    OperatorSelection* fromv;
    OperatorSelection* tov;
};

#endif

```

```

// file          edge.c
// description:   Implementation of Edge class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * this file was made specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last Change made:  October 1, 1990
 */

#include "dfd_defs.h"
#include "edge.h"
#include "istring.h"
#include "sldfdspline.h"
#include "sloperator.h"
#include "slttext.h"
#include <InterViews/Std/string.h>

/*
 * Implementation of class storing the actual data flow and the labels
 * associated with its input and output operators.
 */

Edge::Edge(DFDSplineSelection* f) {
    flow = f;
    fromv = nil;
    tov = nil;
}

char* Edge::GetEdgeLabel() {
    TextSelection* ts = flow->GetTextSelection();
    char* result;
    if (ts != nil) {
        int len;
        char* tmp = ts->GetOriginal(len);
        result = new char[len+1];
        strncpy(result,tmp,len);
        result[len] = '\0';
    }
    else {
        result = ID_TKN;
    }
    return RemoveBadChars(result);
}

char* Edge::GetEdgeLatency() {
    TextSelection* ts = flow->GetLatencySelection();
    char* result = nil;
    if (ts != nil) {
        int len;
        char* tmp = ts->GetOriginal(len);
    }

```

```

        result = new char[len+1];
        strncpy(result,tmp,len);
        result[len] = '\0';
    }
    return result;
}

char* Edge::GetFromVertexLabel() {
    char* result;
    if (fromv != nil) {
        TextSelection* ts = fromv->GetTextSelection();
        if (ts != nil) {
            int len;
            char* tmp = ts->GetOriginal(len);
            result = new char[len+1];
            strncpy(result,tmp,len);
            result[len] = '\0';
        }
        else {
            result = ID_TKN;
        }
    }
    else {
        result = EXT_TKN;
    }
    return result;
}

char* Edge::GetToVertexLabel() {
    char* result;
    if (tov != nil) {
        TextSelection* ts = tov->GetTextSelection();
        if (ts != nil) {
            int len;
            char* tmp = ts->GetOriginal(len);
            result = new char[len+1];
            strncpy(result,tmp,len);
            result[len] = '\0';
        }
        else {
            result = ID_TKN;
        }
    }
    else {
        result = EXT_TKN;
    }
    return result;
}

```

```

// file          edgelist.h
// description:   Class description of EdgeList class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * This header file was created specifically for the graphic editor.
 * This header file was created specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 1, 1990
 */

#ifndef edgelist_h
#define edgelist_h

#include "list.h"

// declare imported classes

class Edge;
class DFDSplineSelection;

// This class defines a node to be contained in the edge list

class EdgeNode : public BaseNode {
public:
    EdgeNode(Edge* e) { edge = e; }
    boolean SameValueAs(void* e) { return edge == e; },
    Edge* GetSelection() { return edge; }

protected:
    Edge* edge;    // points to an edge in the graph
};

// This class defines a list of edges with their corresponding input
// and output operators

class EdgeList : public BaseList {
public:
    EdgeNode* First();
    EdgeNode* Last();
    EdgeNode* Prev();
    EdgeNode* Next();
    EdgeNode* GetCur();
    boolean AtEnd();
    EdgeNode* Index(int);
    boolean FindSpline(DFDSplineSelection*);
};

inline EdgeNode* EdgeList::First() {
    return (EdgeNode*) BaseList::First();
}

```



```

inline EdgeNode* EdgeList::Last() {
    return (EdgeNode*) BaseList::Last();
}

inline EdgeNode* EdgeList::Prev() {
    return (EdgeNode*) BaseList::Prev();
}

inline EdgeNode* EdgeList::Next() {
    return (EdgeNode*) BaseList::Next();
}

inline EdgeNode* EdgeList::GetCur() {
    return (EdgeNode*) BaseList::GetCur();
}

inline EdgeNode* EdgeList::Index(int index) {
    return (EdgeNode*) BaseList::Index(index);
}

inline boolean EdgeList::AtEnd() {
    return BaseList::AtEnd();
}

#endif

```

```

// file      edgelist.c
// description: Implementation of EdgeList class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * This file was made specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 2, 1990
 */

/* Implementation of the list of edges class.  This class is used to
 * write the list of edges to the graph file
 */

#include "edge.h"
#include "edgelist.h"
#include "sldfdspline.h"
#include <InterViews/defs.h>

boolean EdgeList::FindSpline(DFDSplineSelection* dss) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetEdge() == dss) {
            return true;
        }
    }
    return false;
}

```

```

// file      editor.h
// description: Class description of Editor class.

// $Header: editor.h,v 1.12 89/10/09 14:47:57 linton Exp $
// declares class Editor.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add declaration of function ResetMessage.
 * Add class variable inter to store idraw's interactor in order to
 * change cursor when needed.
 * Add declaration of MakeFilename and HandleMET.
 * Remove declaration of unneeded functions.
 * Change name of HandleAnnotate to HandleSpecify to be consistent.
 * Add declaration of HandleLatency.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 21, 1990
 */

#ifndef editor_h
#define editor_h

#include <InterViews/defs.h>

// Declare imported types.

class ChangeNode;
class Chooser;
class Confirmer;
class Drawing;
class DrawingView;
class Event;
class Selector;
class History;
class IBrush;
class IColor;
class IFont;
class IPattern;
class Interactor;
class Messenger;
class Namer;
class Painter;
class RubberEllipse;
class RubberLine;
class RubberRect;
class State;

// An Editor lets the user perform a drawing or editing operation on
// a Drawing.

class Editor {

```

```

public:

    Editor(Interactor*);
    ~Editor();

    void SetDrawing(Drawing*);
    void SetDrawingView(DrawingView*);
    void SetState(State*);
    void SetDirectory(char*);

    void HandleSelect(Event&);
    void HandleMove(Event&);
    void HandleScale(Event&);
    void HandleModify(Event&);
    void HandleMagnify(Event&);
    void HandleSpecify(Event&);
    void HandleStreams(Event&);
    void HandleConstraints(Event&);
    void HandleDecompose(Event&);
    void HandleText(Event&);
    void HandleLabel(Event&);
    void HandleMET(Event&);
    void HandleLatency(Event&);
    // void HandleLine(Event&);
    void HandleBSpline(Event&);
    void HandleEllipse(Event&);

    /* ***** Start of Commented Out Code *****
    void HandleStretch(Event&);
    void HandleRotate(Event&);
    void HandleMultiLine(Event&);
    void HandleRect(Event&);
    void HandlePolygon(Event&);
    void HandleClosedBSpline(Event&);
    ***** End of Commented Out Code ***** */

    void New();
    void Revert();
    void Open(const char*);
    void Open();
    void Save();
    void SaveAs();
    void Print();
    void Quit(Event&);
    void Checkpoint();

    void Undo();
    void Redo();
    void Cut();
    void Copy();
    void Paste();
    void Duplicate();

```

```

void Delete();
void SelectAll();

/* ***** Start of Commented Out Code *****
void FlipHorizontal();
void FlipVertical();
void _90Clockwise();
void _90CounterCW();
void PreciseMove();
void PreciseScale();
void PreciseRotate();

void Group();
void Ungroup();
void BringToFront();
void SendToBack();
void NumberOfGraphics();
***** End of Commented Out Code ***** */

void SetBrush(IBrush*);
void SetFgColor(IColor*);
void SetBgColor(IColor*);
void SetFont(IFont*);
void SetPattern(IPattern*);

void AlignLeftSides();
void AlignRightSides();
void AlignBottoms();
void AlignTops();
void AlignVertCenters();
void AlignHorizCenters();
void AlignCenters();
void AlignLeftToRight();
void AlignRightToLeft();
void AlignBottomToTop();
void AlignTopToBottom();
void AlignToGrid();

void Reduce();
void Enlarge();
void ReduceToFit();
void NormalSize();
void CenterPage();
void RedrawPage();
void GriddingOnOff();
void GridVisibleInvisible();
void GridSpacing();
void Orientation();
void ShowVersion();
void ResetMessage(const char*);

```

protected:

```

const char* MakeFilename(const char*);
const char* MakeSpecFilename(const char*);
void Do(ChangeNode*);
void InputVertices(Event&, Coord*&, Coord*&, int&);
RubberLine* NewRubberLineOrAxis(Event&);
RubberEllipse* NewRubberEllipseOrCircle(Event&);
RubberRect* NewRubberRectOrSquare(Event&);
boolean OfferToSave();
void Reset(const char*, const char*);

History* history;           // carries out and logs changes made
                             // to drawing
Messenger* numberofdialog; // displays how many graphics the
                             // drawing has
Selector* opendialog;       // prompts for name of a drawing to open
Confirmer* overwritdialog; // confirms whether to overwrite a file
Namer* precmovedialog;      // prompts for X and Y movement in points
Namer* precrottdialog;      // prompts for rotation in degrees
Namer* precscaldialog;      // prompts for X and Y scaling
Namer* printdialog;         // prompts for print command
Messenger* readonlydialog;  // tells user drawing is readonly
Confirmer* reverttdialog;   // confirms whether to revert from a file
Selector* saveasdialog;     // prompts for name to save drawing as
Confirmer* savecurdialog;   // confirms whether to save current drawing
Namer* spacingdialog;       // prompts for grid spacing in points
Messenger* versiondialog;   // displays idraw's version level
                             // and author
Chooser* decomposedialog;   // displays decomposition choices
Messenger* nolabeldialog;   // tells user that cannot decompose because
                             // operator chosen has no label

Drawing* drawing;           // performs operations on drawing
DrawingView* drawingview;   // displays drawing
State* state;               // stores Graphic and nonGraphic attributes
char* dir;                  // directory where prototypes are stored
Interactor* inter;          // store the interactor (idraw) for editor
                             // in order to change cursor for Annotate

};

#endif

```

```

// file          editor.c
// description:   Implementation of Editor class.

// $Header: editor.c,v 1.22 89/10/25 18:08:41 interran Exp $
// implements class Editor.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Use a selector instead of finder to display all prototypes available for
 * editing.
 * Add new variable to store name of directory of prototypes.
 * Remove the functions HandleStretch, HandleRotate, HandleMultiLine,
 * HandleRect, HandlePolygon, HandleCloseBSpline, FlipHorizontal,
 * FlipVertical, _90ClockWise, _90CounterCW, PreciseMove, PreciseScale,
 * PreciseRotate, Group, Ungroup, BringToFront, SendToBack, and
 * NumberOfGraphics because their corresponding commands were removed.
 * Changed name of function HandleReshape to HandleModify to be consistent
 * with changing the name of the tool from Reshape to Modify.
 * Added code for drawing line to find out which operators the line is
 * attached to. We need that information for the data flow diagram.
 * Added code for drawing spline to find out which operators the spline is
 * attached to or if it is a self loop. We need that information for the
 * data flow diagram.
 * Commented out code for the former version of HandleEllipse so that it
 * may be used later if needed.
 * Created new HandleEllipse to draw an ellipse of fixed radius.
 * Throughout all of the functions, the file name of the drawing is derived
 * from the prototype name concatenated with ".graph". Idraw used file
 * names given explicitly by the user.
 * Add function MakeFilename to construct the file name of the drawing from
 * the prototype name.
 * Add function ResetMessage to change the message block in the editor.
 * Add function HandleMET to add the maximum execution time of an
 * operator.
 * Update the PSDL specification for the drawing when saving or changing
 * the prototype name.
 * Comment out all references to lines because splines will be used for
 * lines now.
 * Change name of HandleAnnotate to HandleSpecify to be consistent with
 * what user sees.
 * Add function HandleLatency to add latency of data flow to drawing.
 *
 * Changes made by:   Mary Ann Cummings
 * Last Change made:  October 3, 1990
 */

#include "dfdclasses.h"
#include "dfd_defs.h"
#include "dialogbox.h"
#include "drawing.h"
#include "drawingview.h"
#include "editor.h"

```

```

#include "history.h"
#include "idraw.h"
#include "istring.h"
#include "listchange.h"
#include "listselectn.h"
#include "rubbands.h"
#include "selecter.h"
#include "selection.h"
#include "slellipses.h"
#include "sllines.h"
#include "slpolygons.h"
#include "slsplines.h"
#include "slttext.h"
#include "state.h"
#include "textedit.h"
#include "version.h"
#include <InterViews/cursor.h>
#include <InterViews/event.h>
#include <InterViews/transformer.h>
#include <InterViews/Graphic/util.h>
#include <sys/param.h>
#include <bstring.h>
#include <InterViews/Std/stdio.h>
#include <string.h>

// design database function used to find all prototypes in directory

void find_prototype_names(char**, char*, const char*);

// Editor creates its history and dialog boxes.

Editor::Editor (Interactor* i) {
    history = new History(i);
    numberofdialog = new Messenger(i);
    opendialog = new Selector(i, "Select prototype to edit:", Center);
    overwrittenialog= new Confirmer(i, "already exists; overwrite?");
    precmovedialog =new Namer(i,"Enter X and Y movement in printer's
points:");
    precrottdialog = new Namer(i, "Enter rotation in degrees:");
    precscaldialog = new Namer(i, "Enter X and Y scaling:");
    printdialog = new Namer(i, "Enter print command:");
    readonlydialog = new Messenger(i, "Drawing is readonly.");
    reverttdialog = new Confirmer(i, "Really revert to original?");
    saveasdialog = new Selector(i, "Select prototype to save:", Center);
    savecurdialog = new Confirmer(i, "Save current drawing?");
    spacingdialog = new Namer(i, "Enter grid spacing in printer's
points:");
    versiondialog = new Messenger(i, version);
    decomposedialog = new Chooser(i, "Choose decomposition type:",
    "Graphic Editor", "      Ada      ",
    "      Search      ");

    nolabeldialog =

```



```

        new Messenger(i, "Operator has no label, cannot decompose.");

        drawing = nil;
        drawingview = nil;
        state = nil;
        inter = i;
    }

    // ~Editor frees storage allocated for its history and dialog boxes.

Editor::~Editor () {
    delete history;
    delete numberofdialog;
    delete opendialog;
    delete overwrittenialog;
    delete precmovedialog;
    delete precrottdialog;
    delete precscaledialog;
    delete printdialog;
    delete readonlydialog;
    delete revertdialog;
    delete saveasdialog;
    delete savecurdialog;
    delete spacingdialog;
    delete versiondialog;
}

// Define access functions to set members' values. Only Idraw sets
// their values.

void Editor::SetDrawing (Drawing* d) {
    drawing = d;
}

void Editor::SetDrawingView (DrawingView* dv) {
    drawingview = dv;
}

void Editor::SetState (State* s) {
    state = s;
}

// set the directory of prototypes to the given character string

void Editor::SetDirectory(char* d) {
    dir = new char [MAXPATHLEN + 1];
    strcpy(dir,d);
}

// HandleSelect lets the user pick a Selection if one's under the
// mouse, otherwise it lets the user manipulate a rubber rectangle to
// enclose the Selection he wants to pick. HandleSelect clears all

```

```

// previous Selections unless the user holds down the shift key to
// extend the Selections being made.

void Editor::HandleSelect (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    if (!e.shift) {    // replacing previous Selections
        drawingview->EraseHandles();
        if (pick != nil) {
            drawing->Select(pick);
        } else {
            RubberRect* rubberrect =
            new RubberRect(nil, nil, e.x, e.y, e.x, e.y);
            drawingview->Manipulate(e, rubberrect, UpEvent, false);
            Coord l, b, r, t;
            rubberrect->GetCurrent(l, b, r, t);
            delete rubberrect;

            SelectionList* picklist= drawing->PickSelectionsWithin(l, b, r, t);
            drawing->Select(picklist);
            delete picklist;
        }
    } else {    // extending Selections
        if (pick != nil) {
            drawingview->ErasePickedHandles(pick);
            drawing->Extend(pick);
        } else {
            RubberRect* rubberrect =
            new RubberRect(nil, nil, e.x, e.y, e.x, e.y);
            drawingview->Manipulate(e, rubberrect, UpEvent, false);
            Coord l, b, r, t;
            rubberrect->GetCurrent(l, b, r, t);
            delete rubberrect;

            SelectionList* picklist= drawing->PickSelectionsWithin(l, b, r, t);
            drawingview->ErasePickedHandles(picklist);
            drawing->Extend(picklist);
            delete picklist;
        }
    }
    drawingview->DrawHandles();
}

// HandleMove lets the user manipulate a sliding rectangle enclosing
// the Selections and moves them the same way when the user releases
// the button.

void Editor::HandleMove (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    if (pick != nil) {
        drawingview->EraseUngraspedHandles(pick);
        drawing->Grasp(pick);
        drawingview->DrawHandles();
    }
}

```

```

Coord l, b, r, t;
drawing->GetBox(l, b, r, t);
state->Constrain(e.x, e.y);
SlidingRect* slidingrect =
    new SlidingRect(nil, nil, l, b, r, t, e.x, e.y);
drawingview->Manipulate(e, slidingrect, UpEvent);
Coord nl, nb, nr, nt;
slidingrect->GetCurrent(nl, nb, nr, nt);
delete slidingrect;

if (nl != l || nb != b) {
    float x0, y0, x1, y1;
    Transformer t;
    drawing->GetPictureTT(t);
    t.InvTransform(float(l), float(b), x0, y0);
    t.InvTransform(float(nl), float(nb), x1, y1);
    Do(new MoveChange(drawing, drawingview, x1 - x0, y1 - y0));
}
}

// HandleScale lets the user manipulate a scaling rectangle enclosing
// the picked Selection and scales the Selections to the new scale
// when the user releases the button.

void Editor::HandleScale (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    if (pick != nil) {
        drawingview->EraseUngraspedHandles(pick);
        drawing->Grasp(pick);
        drawingview->DrawHandles();

        float l, b, r, t;
        pick->GetBounds(l, b, r, t);
        float cx, cy;
        pick->GetCenter(cx, cy);
        ScalingRect* scalingrect =
            new ScalingRect(nil, nil, round(l), round(b), round(r), round(t),
                round(cx), round(cy));
        drawingview->Manipulate(e, scalingrect, UpEvent);
        float scale = scalingrect->CurrentScaling();
        delete scalingrect;

        if (scale != 0) {
            Do(new ScaleChange(drawing, drawingview, scale, scale));
        }
    }
}

// The following is not needed for a data flow diagram

/* ***** Start of Commented Out Code *****

```

```

// HandleStretch lets the user manipulate a stretching rectangle
// enclosing the picked Selection and stretches the Selections the
// same way when the user releases the button.

void Editor::HandleStretch (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    if (pick != nil) {
        drawingview->EraseUngraspedHandles(pick);
        drawing->Grasp(pick);
        drawingview->DrawHandles();

        float l, b, r, t;
        pick->GetBounds(l, b, r, t);
        IStretchingRect* istretchingrect = new
            IStretchingRect(nil, nil, round(l), round(b), round(r), round(t));
        drawingview->Manipulate(e, istretchingrect, UpEvent);
        float stretch = istretchingrect->CurrentStretching();
        Alignment side = istretchingrect->CurrentSide(drawing->GetLandscape());
        delete istretchingrect;

        if (stretch != 0) {
            Do(new StretchChange(drawing, drawingview, stretch, side));
        }
    }
}

// HandleRotate lets the user manipulate a rotating rectangle
// enclosing the picked Selection and rotates the Selections the same
// way when the user releases the button.

void Editor::HandleRotate (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    if (pick != nil) {
        drawingview->EraseUngraspedHandles(pick);
        drawing->Grasp(pick);
        drawingview->DrawHandles();

        Coord l, b, r, t;
        pick->GetBox(l, b, r, t);
        float cx, cy;
        pick->GetCenter(cx, cy);
        state->Constrain(e.x, e.y);
        RotatingRect* rotatingrect =
            new RotatingRect(nil, nil, l, b, r, t, round(cx), round(cy),
                e.x, e.y);
        drawingview->Manipulate(e, rotatingrect, UpEvent);
        float angle = rotatingrect->CurrentAngle();
        delete rotatingrect;

        if (angle != 0) {
            Do(new RotateChange(drawing, drawingview, angle));
        }
    }
}

```

```

    }
    }
}

***** End of Commented Out Code ***** */

// HandleModify lets the user modify an already existing Selection
// and replaces the Selection with the modified Selection. Text
// Selections "modify" themselves using different code below.

void Editor::HandleModify (Event& e) {
    Selection* pick = drawing->PickSelectionShapedBy(e.x, e.y);
    if (pick != nil) {
        drawingview->EraseHandles();
        drawing->Select(pick);
        drawingview->DrawHandles();
        Selection* modifiedpick = nil;
        ClassId cid = pick->GetClassId();
        if (cid == LABEL_OP || cid == LABEL_DF || cid == COMMENT ||
            cid == LAT_DF || cid == MET_OP || cid == LABEL_SL) {
            printf("in modify, is text\n");
            printf("cid is %u\n", cid);
            int len;
            const char* text = ((TextSelection*) pick)->GetOriginal(len);
            TextEdit* textedit = new TextEdit(text, len);
            drawingview->EraseHandles();
            drawingview->Edit(e, textedit, pick);
            text = textedit->GetText(len);
            printf("modified text is %s\n", text);
            modifiedpick = new TextSelection(cid, text, len, pick);
            delete textedit;
        } else {
            printf("in modify, is not text\n");
            Rubberband* shape = pick->CreateShape(e.x, e.y);
            drawingview->Manipulate(e, shape, UpEvent);
            modifiedpick = pick->GetReshapedCopy();
        }
        if (modifiedpick != nil) {
            printf("before ReplaceChange\n");
            Do(new ReplaceChange(drawing, drawingview, pick, modifiedpick));
            printf("after ReplaceChange\n");
            drawingview->Draw();
        }
    }
}

// HandleMagnify lets the user manipulate a rubber rectangle and
// expands the given area to fill the view.

void Editor::HandleMagnify (Event& e) {
    RubberRect* rubberrect = NewRubberRectOrSquare(e);
    drawingview->Manipulate(e, rubberrect, UpEvent, false);
    Coord fx, fy;

```

```

        rubberrect->GetCurrent(fx, fy, e.x, e.y);
        delete rubberrect;

        drawingview->Magnify(fx, fy, e.x, e.y);
    }

    // HandleSpecify lets the user select an operator and a syntax directed
    // editor will open with the PSDL that represents that operator

void Editor::HandleSpecify(Event& e) {
    boolean have_operator;
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    drawingview->EraseHandles();
    if (pick != nil && pick->GetClassId() == LABEL_OP) {
        pick = drawing->FindOperator((TextSelection*) pick);
    }

    if (pick != nil && pick->GetClassId() == OPERATOR) {

//        construct PSDL of an operator's specification

        drawing->Select(pick);
        drawingview->DrawHandles();

//        write operator's psdl to a scratch file

        drawing->WritePSDLForOperator((EllipseSelection*) pick);
        have_operator = true;
    }
    else {

//        collecting PSDL for the whole drawing

        drawing->WritePSDLForDrawing();
        have_operator = false;
    }

//    open a text editor with the psdl file

    int code;
    long status;
    char* filename = MakeTmpFileName(PSDL_FILE);
    inter->SetCursor(hourglass);
    if (fork() == 0) {
        code = execlp("xterm", "xterm", "-T", "specify", "-g",
                     "+500+550", "+sb", "-e", SPECIFICATION_SDE, filename, 0);
        exit(code);
    }
    wait(&status);
    inter->SetCursor(defaultCursor);
    state->SetModifStatus(Modified);
    state->UpdateViews();
}

```

```

    if (have_operator) {
        FILE* fptr = fopen(filename, "r");
        if (fptr != nil) {
            drawing->ReadPSDLForOperator(fptr, (EllipseSelection*) pick);
            fclose(fptr);
        }
    }
    else
        drawing->ReadPSDLForDrawing();
    delete filename;
}

// HandleStreams opens an editor to allow the user to add PSDL streams

void Editor::HandleStreams(Event& e) {
    drawing->WriteStreams();
    int code;
    long status;
    char* filename = MakeTmpFileName(STREAMS_FILE);
    inter->SetCursor(hourglass);
    if (fork() == 0) {
        code = execlp("xterm", "xterm", "-T", "streams", "-g", "+500+550",
                    "+sb", "-e", STREAMS_SDE, filename, 0);
        exit(code);
    }
    wait(&status);
    delete filename;
    inter->SetCursor(defaultCursor);
    state->SetModifStatus(Modified);
    state->UpdateViews();
    drawing->ReadStreams();
}

// HandleConstraints opens an editor to allow the user to add PSDL constraints

void Editor::HandleConstraints(Event& e) {
    drawing->WriteConstraints();
    int code;
    long status;
    char* filename = MakeTmpFileName(CONSTRAINTS_FILE);
    inter->SetCursor(hourglass);
    if (fork() == 0) {
        code = execlp("xterm", "xterm", "-T", "constraints", "-g",
                    "+500+550",
                    "+sb", "-e", CONSTRAINTS_SDE, filename, 0);
        exit(code);
    }
    wait(&status);
    delete filename;
    inter->SetCursor(defaultCursor);
    state->SetModifStatus(Modified);
    state->UpdateViews();
}

```

```

        drawing->ReadConstraints();
    }

    // HandleDecompose gives the user of decomposition types and then
    // processes the choice

void Editor::HandleDecompose(Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    drawingview->EraseHandles();
    if (pick != nil && pick->GetClassId() == LABEL_OP) {
        pick = drawing->FindOperator((TextSelection*) pick);
    }
    if (pick != nil && pick->GetClassId() == OPERATOR) {
        drawing->Select(pick);
        drawingview->DrawHandles();
        char* label = drawing->OperatorLabelIs((EllipseSelection*) pick);
        printf("label is %s\n", label);
        if (label == nil) {
            nolabeldialog->Display();
        }
        else {
            char* fixed_label = RemoveBadChars(label);
            char result = decomposedialog->Choose();
            Save();
            const char* prototype_name = state->GetDrawingName();
            printf("prot name is %s\n", prototype_name);
            switch (result) {
                case 'f':

//                first button pushed, decompose with graphic editor

                char* new_prot_name = new char[strlen(prototype_name)
                                                + strlen(fixed_label) + 2];
                strcpy(new_prot_name, prototype_name);
                strcat(new_prot_name, ".");
                strcat(new_prot_name, fixed_label);
                printf("new prot name is %s\n", new_prot_name);

                int code;
                long status;
                inter->SetCursor(hourglass);
                if (fork() == 0) {
                    code = execlp("graphic_editor", "graphic_editor",
                                "-d", dir, "-p", new_prot_name,
                                "-geometry", "+450-200", 0);
                    exit(code);
                }
                wait(&status);
                char* s_filename = new char[strlen(dir) + strlen(new_
prot_name)
                                                + SPEC_EXT_LEN + 1];
                strcpy(s_filename, dir);

```



```

        strcat(s_filename,new_prot_name);
        strcat(s_filename,SPEC_PSDL_EXT);
        FILE* sptr = fopen(s_filename, "r");
        if (sptr != nil) {
            drawing->ReadPSDLForOperator(sptr,
                                         (EllipseSelection*) pick);
        }
        delete s_filename;
        inter->SetCursor(defaultCursor);
        delete new_prot_name;
        break;
    case 's':
    case 't':

//          second button chosen, write component directly in Ada
//          third button chosen, search for reusable components to
//          match operator's specification

        char* filename = new char[strlen(dir) + strlen(prototype_
name)
                                + strlen(fixed_label) + IMP_EXT_LEN + 2];
        strcpy(filename,dir);
        strcat(filename,prototype_name);
        strcat(filename,".");
        strcat(filename,fixed_label);
        strcat(filename,IMP_PSDL_EXT);
        FILE* fptr = fopen(filename, "w");
        fprintf(fptr, "%s%s\n", IMP_ADA_TKN, label);
        fprintf(fptr, "%s", END_TKN);
        fclose(fptr);
        delete filename;
        break;
    default:
        break;
    }
}

}

// HandleText lets the user type some text and creates a new
// TextSelection when the user finishes typing the text. It must
// clear the selection list because DrawingView will redraw the
// handles obscured by the TextEdit if the list's not empty.

void Editor::HandleText (Event& e) {
    drawingview->EraseHandles();
    drawing->Clear();
    TextEdit* textedit = new TextEdit;
    drawingview->Edit(e, textedit);
    int len;
    const char* text = textedit->GetText(len);

```

```

        if (len > 0) {
            drawing->Select(new TextSelection(text, len, state->GetTextGS()));
            Do(new AddChange(drawing, drawingview));
        }
        delete textedit;
    }

// HandleLabel lets the user pick a Selection and adds text in that se-
// lection.
// The text will then be centered on the selection.

void Editor::HandleLabel (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    drawingview->EraseHandles();
    if (pick != nil) {
        ClassId cid_pick = pick->GetClassId();
        drawing->Select(pick);
        drawingview->DrawHandles();

//        can only add label to one selection at a time

        if (drawing->GetNumberOfGraphics() == 1) {
            TextEdit* textedit = new TextEdit;
            drawingview->Edit(e, textedit);
            int len;
            const char* text = textedit->GetText(len);
            if (len > 0) {

//                determine the type of label that we are using

                ClassId cid_text;
                if (cid_pick == OPERATOR) {
                    cid_text = LABEL_OP;
                }
                else {
                    if (cid_pick == DATAFLOW_SPLINE
//                        || cid_pick == DATAFLOW_LINE
                        ) {
                        cid_text = LABEL_DF;
                    }
                    else {
                        if (cid_pick == SELFLOOP) {
                            cid_text = LABEL_SL;
                        }
                        else {
                            cid_text = COMMENT;
                        }
                    }
                }
                TextSelection* ts = new TextSelection(cid_text, text, len,
                                                    state->GetTextGS());
            }
        }
    }
}

```

```

//          add label to operator list

drawing->LabelAppend(ts, pick);
drawing->Select(ts);
Do(new AddChange(drawing, drawingview));
drawing->Clear();

//          center label to selection

SelectionList* sl = new SelectionList;
sl->Append(new SelectionNode(pick));
sl->Append(new SelectionNode(ts));
drawing->Select(sl);
AlignCenters();
drawingview->EraseHandles();
drawing->Clear();
    }
    delete textedit;
}
}

// HandleMET letes the user select an operator, add the maximum execution
// time associated with that operator. This MET is place on top of the
// operator.

void Editor::HandleMET (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    drawingview->EraseHandles();
    if (pick != nil) {
        if (pick->GetClassId() == LABEL_OP) {
            pick = drawing->FindOperator((TextSelection*) pick);
        }

//          can only add the MET to an operator

        if (pick->GetClassId() == OPERATOR) {
            drawing->Select(pick);
            drawingview->DrawHandles();
            TextEdit* textedit = new TextEdit;
            drawingview->Edit(e, textedit);
            int len;
            const char* text = textedit->GetText(len);
            if (len > 0) {
                TextSelection* ts = new TextSelection(MET_OP, text, len,
                                                    state->GetTextGS());

//          add MET to operator list

                drawing->METAppend(ts, (EllipseSelection*) pick);
                drawing->Select(ts);
                Do(new AddChange(drawing, drawingview));
            }
        }
    }
}

```

```

        drawing->Clear();

//        place MET on top of operator

        SelectionList* sl = new SelectionList;
        sl->Append(new SelectionNode(pick));
        sl->Append(new SelectionNode(ts));
        drawing->Select(sl);
        AlignCenters();
        AlignBottomToTop();
        drawingview->EraseHandles();
        drawing->Clear();
    }
    delete textedit;
}

// HandleLatency letes the user select a data flow, add the latency
// associated with that data flow. This latency is placed on top of the
// data flow.

void Editor::HandleLatency (Event& e) {
    Selection* pick = drawing->PickSelectionIntersecting(e.x, e.y);
    drawingview->EraseHandles();
    if (pick != nil) {

//        can only add the MET to an operator

        if (pick->GetClassId() == DATAFLOW_SPLINE) {
            Selection* oldpick = pick;
            pick = drawing->FindLabel(pick);
            drawing->Select(oldpick);
            drawingview->DrawHandles();
            TextEdit* textedit = new TextEdit;
            drawingview->Edit(e, textedit);
            int len;
            const char* text = textedit->GetText(len);
            if (len > 0) {
                TextSelection* ts = new TextSelection(LAT_DF, text, len,
                                                    state->GetTextGS());

//                add latency to operator list

                drawing->LatencyAppend(ts, (BSplineSelection*) oldpick);
                drawing->Select(ts);
                Do(new AddChange(drawing, drawingview));
                drawing->Clear();

//                place Latency on top of data flow

                SelectionList* sl = new SelectionList;
                sl->Append(new SelectionNode(pick));

```

```

        sl->Append(new SelectionNode(ts));
        drawing->Select(sl);
        AlignCenters();
        AlignBottomToTop();
        drawingview->EraseHandles();
        drawing->Clear();
    }
    delete textedit;
}
}

// Use splines to draw lines also

/* ***** Start of Commented Out Code *****

// HandleLine lets the user manipulate a rubber line and creates
// a LineSelection when the user releases the button.

void Editor::HandleLine (Event& e) {
    drawingview->EraseHandles();
    state->Constrain(e.x, e.y);
    RubberLine* rubberline = NewRubberLineOrAxis(e);
    drawingview->Manipulate(e, rubberline, UpEvent);
    Coord x0, y0, x1, y1;
    rubberline->GetCurrent(x0, y0, x1, y1);
    delete rubberline;

    if (x0 != x1 || y0 != y1) {

//      determine which operator the line is attached to and recompute
//      the line's endpoint to be the intersection of the operator and the
//      line

        EllipseSelection* es0;
        EllipseSelection* es1;
        es0 = drawing->SetEndptsInOperator(x0, y0, x1, y1);
        es1 = drawing->SetEndptsInOperator(x1, y1, x0, y0);

        if (es0 != nil || es1 != nil) {

//          use the arrowhead pattern when drawing the line

            IPattern* temp = state->GetPattern();
            state->SetPattern(state->GetArrowPattern());
            LineSelection* ls =
            new LineSelection(x0, y0, x1, y1, state->GetGraphicGS());
            drawing->DataFlowLineAppend(ls, es0, es1);
            drawing->Select(ls);

//          reset the pattern back to the original pattern, not the arrowhead
//          pattern

```

```

        state->SetPattern(temp);
        Do(new AddChange(drawing, drawingview));
        drawingview->EraseHandles();
    }
}

// The following is not needed for a data flow diagram

// HandleMultiLine lets the user draw a series of connected lines and
// creates a MultiLineSelection when the user presses the middle
// button.

void Editor::HandleMultiLine (Event& e) {
    Coord* x;
    Coord* y;
    int n;

    drawingview->EraseHandles();
    InputVertices(e, x, y, n);

    if (n != 2 || x[0] != x[1] || y[0] != y[1]) {
        drawing->Select(
            new MultiLineSelection(x, y, n, state->GetGraphicGS())
        );
        Do(new AddChange(drawing, drawingview));
    }
}

***** End of Commented Out Code ***** */

// HandleBSpline lets the user draw a series of connected lines and
// creates a BSplineSelection when the user presses the middle button.

void Editor::HandleBSpline (Event& e) {
    Coord* x;
    Coord* y;
    int n;

    drawingview->EraseHandles();
    InputVertices(e, x, y, n);

    if (n != 2 || x[0] != x[1] || y[0] != y[1]) {

//      determine which operators the spline is attached to and recompute
//      the spline's endpoints to be the intersection of the operator and
//      the spline

        EllipseSelection* es0;
        EllipseSelection* esn;
        es0 = drawing->SetEndptsInOperator(x[0], y[0], x[1], y[1]);
        esn = drawing->SetEndptsInOperator(x[n-1], y[n-1], x[n-2], y[n-2]);
    }
}

```

```

        if (es0 != nil || esn != nil) {

//            determine type of spline

            boolean is_stream = false;
            ClassId classid;
            if (es0 == esn) {
                classid = SELFLOOP;
            }
            else {
                classid = DATAFLOW_SPLINE;
                if (es0 != nil && esn != nil) {
                    is_stream = true;
                    drawing->AddStream();
                }
            }

//            use the arrowhead's pattern to draw the line

            IPattern* temp = state->GetPattern();
            state->SetPattern(state->GetArrowPattern());
            BSplineSelection* ss =
                new BSplineSelection(classid, x, y, n, state->GetGraph-
icGS());
            if (is_stream)
                ss->SetStream();

//            append the spline to the operator list

            drawing->DataFlowSplineAppend(ss, es0, esn);
            drawing->Select(ss);

//            reset the pattern to be the original pattern

            state->SetPattern(temp);
            Do(new AddChange(drawing, drawingview));
            drawingview->EraseHandles();
        }
    }

// This is the old HandleEllipse that used a rubber ellipse to draw it

/* ***** Start of Commented Out Code *****

// HandleEllipse lets the user manipulate a rubber ellipse and creates
// an EllipseSelection when the user releases the button.

void Editor::HandleEllipse (Event& e) {
    drawingview->EraseHandles();
    state->Constrain(e.x, e.y);
    RubberEllipse* rubberellipse = NewRubberEllipseOrCircle(e);

```

```

drawingview->Manipulate(e, rubberellipse, UpEvent);
Coord cx, cy, rx, ry;
int xr, yr;
rubberellipse->GetCurrent(cx, cy, rx, ry);
rubberellipse->CurrentRadii(xr, yr);
delete rubberellipse;

if (xr > 0 || yr > 0) {
    drawing->Select(
        new EllipseSelection(cx, cy, xr, yr, state->GetGraphsGS())
    );
    Do(new AddChange(drawing, drawingview));
}
}

***** End of Commented Out Code ***** */

// HandleEllipse draws a circle with a radius of 35 pixels at the position
// of the user's mouse when he clicks the left mouse button

void Editor::HandleEllipse (Event& e) {
    drawingview->EraseHandles();
    state->Constrain(e.x, e.y);
    EllipseSelection* es =
        new EllipseSelection(e.x, e.y, OperatorRadius, OperatorRadius,
                             state->GetGraphicGS());

    drawing->Select(es);
    drawing->OperatorAppend(es);
    Do(new AddChange(drawing, drawingview));
}

// The following is not needed for a data flow diagram

/* ***** Start of Commented Out Code *****

// HandleRect lets the user manipulate a rubber rectangle and creates
// a RectSelection when the user releases the button.

void Editor::HandleRect (Event& e) {
    drawingview->EraseHandles();
    state->Constrain(e.x, e.y);
    RubberRect* rubberrect = NewRubberRectOrSquare(e);
    drawingview->Manipulate(e, rubberrect, UpEvent);
    Coord l, b, r, t;
    rubberrect->GetCurrent(l, b, r, t);
    delete rubberrect;

    if (l != r || b != t) {
        drawing->Select(new RectSelection(l, b, r, t, state->GetGraphicGS()));
        Do(new AddChange(drawing, drawingview));
    }
}
}

```



```

// HandlePolygon lets the user draw a series of connected lines and
// creates a PolygonSelection when the user presses the middle button.

void Editor::HandlePolygon (Event& e) {
    Coord* x;
    Coord* y;
    int n;

    drawingview->EraseHandles();
    InputVertices(e, x, y, n);

    if (n != 2 || x[0] != x[1] || y[0] != y[1]) {
        drawing->Select(new PolygonSelection(x, y, n, state->GetGraphicGS()));
        Do(new AddChange(drawing, drawingview));
    }
}

// HandleClosedBSpline lets the user draw a series of connected lines
// and creates a ClosedBSplineSelection when the user presses the
// middle button.

void Editor::HandleClosedBSpline (Event& e) {
    Coord* x;
    Coord* y;
    int n;

    drawingview->EraseHandles();
    InputVertices(e, x, y, n);

    if (n != 2 || x[0] != x[1] || y[0] != y[1]) {
        drawing->Select(
            new ClosedBSplineSelection(x, y, n, state->GetGraphicGS())
        );
        Do(new AddChange(drawing, drawingview));
    }
}

***** End of Commented Out Code ***** */

// New offers to write an unsaved drawing and creates a new empty
// drawing if the save succeeds or the user refuses the offer.

void Editor::New () {
    boolean successful = OfferToSave();
    if (successful) {
        drawing->ClearPicture();
        Reset(nil, nil);
    }
}

// Revert rereads the drawing from its file. It asks for confirmation
// before reverting an unsaved drawing.

```

```

void Editor::Revert () {
    const char* prototype_name = state->GetDrawingName();
    if (prototype_name != nil) {
        char response = revertdialog->Confirm();
        if (response == 'y') {
            const char* filename = MakeFilename(prototype_name);
            boolean successful = drawing->ReadPicture(filename, state);
            if (successful) {
                drawing->ReadDFDFiles(dir, prototype_name);
                Reset(filename, prototype_name);
            }
            else {
                savecurdialog->
                SetWarning("couldn't revert! (file nonexistent?)");
                Open();
            }
        }
    }
}

// Open reads a drawing from a file whose filename is created by the given
// prototype name. If it fails, it calls the interactive Open to ask the
// user to type another name.

void Editor::Open (const char* prototype_name) {
    const char* filename;
    filename = MakeFilename(prototype_name);
    drawing->ReadPicture(filename, state);
    drawing->ReadDFDFiles(dir, prototype_name);
    const char* spec_filename = MakeSpecFilename(prototype_name);
    if (!drawing->Exists(spec_filename)) {
        drawing->UpdatePSDLSpec(prototype_name);
    }
    Reset(filename, prototype_name);
}

// Open prompts for a prototype name. It then appends ".graph" to that
// name to make the proper file name and reads a drawing from that file.
// It offers to save an unsaved drawing and it keeps trying to read a
// drawing until it succeeds or the user cancels the command.

void Editor::Open () {
    boolean successful = OfferToSave();
    if (successful) {
        const char* prototype_name = nil;
        const char* filename;
        char* prototype_array[MAXPROTOTYPES];
        find_prototype_names(prototype_array, dir, "edit");
        opendialog->Insert(prototype_array);
        for (;;) {
            prototype_name = opendialog->Select();
            if (prototype_name == nil)

```

```

        break;
    else {
        filename = MakeFilename(prototype_name);
        boolean successful = drawing->ReadPicture(filename, state);
        drawing->ReadDFDFiles(dir, prototype_name);
        const char* spec_filename =
            MakeSpecFilename(prototype_name);
        if (!drawing->Exists(spec_filename)) {
            drawing->UpdatePSDLSpec(prototype_name);
        }
        Reset(filename, prototype_name);
        break;
    }
}
}
}

```

// Save writes the drawing to the file it was read from unless there's
// no file or it can't write the drawing to that file, in which case
// it hands the job off to SaveAs.

```

void Editor::Save () {
    const char* prototype_name = state->GetDrawingName();
    if (prototype_name == nil) {
        SaveAs();
    }
    else if (state->GetModifStatus() == ReadOnly) {
        saveasdialog->SetErrorTitle("Can't save in read-only file!");
        SaveAs();
    }
    else {
        const char* filename = MakeFilename(prototype_name);
        boolean successful = drawing->WritePicture(filename, state);
        if (successful) {
            state->SetModifStatus(Unmodified);
            state->UpdateViews();
            drawing->WriteDFDFiles(dir, prototype_name);
        }
        else {
            saveasdialog->SetErrorTitle("Couldn't save!");
            SaveAs();
        }
    }
}
}

```

// SaveAs prompts for a prototype name. It then uses that name to
// determine the file name and writes the drawing to that file.
// It asks for confirmation before overwriting an already existing
// file and it keeps trying to write the drawing until it succeeds or
// the user cancels the command.

```

void Editor::SaveAs () {

```

```

const char* prototype_name = nil;
char* prototype_array[MAXPROTOTYPES];
for (;;) {
    find_prototype_names(prototype_array, dir, "edit");
    saveasdialog->Insert(prototype_array);
    prototype_name = saveasdialog->Select();
    if (prototype_name == nil)
        break;
    else {
        const char* filename = MakeFilename(prototype_name);
        if (drawing->Exists(filename)) {
            overwrittenialog->SetWarning("a drawing named ", proto-
type_name);
            char response = overwrittenialog->Confirm();
            if (response != 'y')
                break;
        }
        boolean successful = drawing->WritePicture(filename, state);
        if (successful) {
            state->SetDrawingName(prototype_name);
            state->SetModifStatus(Unmodified);
            state->UpdateViews();
            drawing->UpdatePSDLSpec(prototype_name);
            drawing->WriteDFDFiles(dir, prototype_name);
            break;
        }
        else
            saveasdialog->SetErrorTitle("Couldn't save!");
    }
    saveasdialog->SetErrorTitle("");
}

// Print prompts for a print command and writes the drawing through a
// pipe to that command's standard input. It keeps trying to print
// the drawing until it succeeds or the user cancels the command.

void Editor::Print () {
    if (state->GetModifStatus() == Modified) {
        savecurdialog->SetWarning("a broken pipe signal won't be caught");
    }
    boolean successful = OfferToSave();
    if (successful) {
        char* cmd = nil;
        for (;;) {
            delete cmd;
            cmd = printdialog->Edit(nil);
            if (cmd == nil) {
                break;
            }
            boolean successful = drawing->PrintPicture(cmd, state);
            if (successful) {

```

```

        break;
    } else {
        printdialog->SetWarning("couldn't execute ", cmd);
    }
}
delete cmd;
}

// Skew comments/code ratio to work around cpp bug
// ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
// ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

// Quit offers to save an unsaved drawing and tells Idraw to quit
// running if the save succeeds or the user refuses the offer.

void Editor::Quit (Event& e) {
    boolean successful = OfferToSave();
    if (successful) {
        e.target = nil;
    }
}

// Checkpoint writes an unsaved drawing to a temporary filename. The
// program currently calls Checkpoint only when an X error occurs.

void Editor::Checkpoint () {
    if (state->GetModifStatus() == Modified) {
        char* path = tempnam("./", "idraw");
        boolean successful = drawing->WritePicture(path, state);
        if (successful) {
            fprintf(stderr, "saved drawing as \"%s\"\\n", path);
        } else {
            fprintf(stderr, "sorry, couldn't save drawing as \"%s\"\\n", path);
        }
        delete path;
    } else {
        fprintf(stderr, "drawing was unmodified, didn't save it\\n");
    }
}

// Undo undoes the last change made to the drawing. Undo does nothing
// if all stored changes have been undone.

void Editor::Undo () {
    history->Undo();
}

// Redo redoes the last undone change made to the drawing, i.e., it
// undoes an Undo. Redo does nothing if it follows a Do.

void Editor::Redo () {

```

```

        history->Redo();
    }

    // Cut removes the Selections and writes them to the clipboard file,
    // overwriting whatever was there previously.

    void Editor::Cut () {
        Do(new CutChange(drawing, drawingview));
    }

    // Copy copies the Selections and writes them to the clipboard file,
    // overwriting whatever was there previously.

    void Editor::Copy () {
        Do(new CopyChange(drawing, drawingview));
    }

    // Paste reads new Selections from the clipboard file and appends them
    // to the drawing.

    void Editor::Paste () {
        Do(new PasteChange(drawing, drawingview, state));
    }

    // Duplicate duplicates the Selections and appends the new Selections
    // to the drawing.

    void Editor::Duplicate () {
        Do(new DuplicateChange(drawing, drawingview));
    }

    // Delete deletes all of the Selections.

    void Editor::Delete () {
        Do(new DeleteChange(drawing, drawingview));
    }

    // SelectAll selects all of the Selections in the drawing.

    void Editor::SelectAll () {
        drawing->SelectAll();
        drawingview->DrawHandles();
    }

    // The following was removed because it is not needed for a Data Flow
    // diagram

    /* ***** Start of Commented Out Code *****

    // FlipHorizontal flips the Selections horizontally by scaling them by
    // -1 along the x axis about their centers.

```

```

void Editor::FlipHorizontal () {
    Do(new ScaleChange(drawing, drawingview, -1, 1));
}

// FlipVertical flips the Selections vertically by scaling them by -1
// along the y axis about their centers.

void Editor::FlipVertical () {
    Do(new ScaleChange(drawing, drawingview, 1, -1));
}

// _90Clockwise rotates the Selections 90 degrees clockwise about
// their centers.

void Editor::_90Clockwise () {
    Do(new RotateChange(drawing, drawingview, -90.));
}

// _90CounterCW rotates the Selections 90 degrees counter-clockwise
// about their centers.

void Editor::_90CounterCW () {
    Do(new RotateChange(drawing, drawingview, 90.));
}

// PreciseMove prompts the user for the x and y movement and moves the
// Selections that much from their original places in units of points.
// If we didn't use points as units, the user wouldn't be able to move
// a Selection one grid spacing by typing "8 0".

void Editor::PreciseMove () {
    char* movement = nil;
    for (;;) {
        delete movement;
        movement = precmovedialog->Edit(nil);
        if (movement == nil) {
            break;
        }
        float xdisp, ydisp;
        if (sscanf(movement, "%f %f", &xdisp, &ydisp) == 2) {
            if (xdisp != 0 || ydisp != 0) {
                xdisp *= points;
                ydisp *= points;
                Do(new MoveChange(drawing, drawingview, xdisp, ydisp));
            }
            break;
        } else {
            precmovedialog->SetWarning("couldn't parse ", movement);
        }
        delete movement;
    }
}

```

```
// PreciseScale prompts the user for the x and y scales and scales the
// Selections that much about their centers.
```

```
void Editor::PreciseScale () {
    char* scaling = nil;
    for (;;) {
        delete scaling;
        scaling = precscaldialog->Edit(nil);
        if (scaling == nil) {
            break;
        }
        float xscale, yscale;
        if (sscanf(scaling, "%f %f", &xscale, &yscale) == 2) {
            if (xscale != 0 && yscale != 0) {
                Do(new ScaleChange(drawing, drawingview, xscale, yscale));
            }
            break;
        } else {
            precscaldialog->SetWarning("couldn't parse ", scaling);
        }
    }
    delete scaling;
}
```

```
// PreciseRotate prompts the user for the angle and rotates the
// Selections that many degrees about their centers.
```

```
void Editor::PreciseRotate () {
    char* rotation = nil;
    for (;;) {
        delete rotation;
        rotation = precrottdialog->Edit(nil);
        if (rotation == nil) {
            break;
        }
        float angle;
        if (sscanf(rotation, "%f", &angle) == 1) {
            if (angle != 0) {
                Do(new RotateChange(drawing, drawingview, angle));
            }
            break;
        } else {
            precrottdialog->SetWarning("couldn't parse ", rotation);
        }
    }
    delete rotation;
}
```

```
// Group groups the Selections together.
```

```
void Editor::Group () {
```



```

        Do(new GroupChange(drawing, drawingview));
    }

    // Ungroup ungroups each PictSelection into its component Selections.

    void Editor::Ungroup () {
        Do(new UngroupChange(drawing, drawingview));
    }

    // BringToFront brings the Selections to the front of the drawing.

    void Editor::BringToFront () {
        Do(new BringToFrontChange(drawing, drawingview));
    }

    // SendToBack sends the Selections to the back of the drawing.

    void Editor::SendToBack () {
        Do(new SendToBackChange(drawing, drawingview));
    }

    // NumberOfGraphics counts the number of graphics in the drawing and
    // displays the count.

    void Editor::NumberOfGraphics () {
        int num = drawing->GetNumberOfGraphics();
        if (num == 1) {
            numberofdialog->SetMessage("The selections contain 1 graphic.");
        } else {
            char buf[50];
            sprintf(buf, "The selections contain %d graphics.", num);
            numberofdialog->SetMessage(buf);
        }
        numberofdialog->Display();
    }

    ***** End of Commented Out Code ***** */

    // SetBrush sets the Selections' brush and updates the views to
    // display the new current brush.

    void Editor::SetBrush (IBrush* brush) {
        state->SetBrush(brush);
        state->UpdateViews();
        Do(new SetBrushChange(drawing, drawingview, brush));
    }

    // Skew comments/code ratio to work around cpp bug
    ;;;;;;;;;;
    ;;;;;;;;;;
    ;;;;;;;;;;
    ;;;;;;;;;;
    ;;;;;;;;;;

```

```

// SetFgColor sets the Selections' foreground color and updates the
// views to display the new current foreground color.

void Editor::SetFgColor (IColor* fg) {
    state->SetFgColor(fg);
    state->UpdateViews();
    Do(new SetFgColorChange(drawing, drawingview, fg));
}

// SetBgColor sets the Selections' background color and updates the
// views to display the new current background color.

void Editor::SetBgColor (IColor* bg) {
    state->SetBgColor(bg);
    state->UpdateViews();
    Do(new SetBgColorChange(drawing, drawingview, bg));
}

// SetFont sets the Selections' font and updates the views to display
// the new current font.

void Editor::SetFont (IFont* font) {
    state->SetFont(font);
    state->UpdateViews();
    Do(new SetFontChange(drawing, drawingview, font));
}

// SetPattern sets the Selections' pattern and updates the views to
// display the new current pattern.

void Editor::SetPattern (IPattern* pattern) {
    state->SetPattern(pattern);
    state->UpdateViews();
    Do(new SetPatternChange(drawing, drawingview, pattern));
}

// AlignLeftSides aligns the rest of the Selections's left sides with
// the first Selection's left side.

void Editor::AlignLeftSides () {
    Do(new AlignChange(drawing, drawingview, Left, Left));
}

// AlignRightSides aligns the rest of the Selections' right sides with
// the first Selection's right side.

void Editor::AlignRightSides () {
    Do(new AlignChange(drawing, drawingview, Right, Right));
}

// AlignBottomSides aligns the rest of the Selections' bottom sides
// with the first Selection's bottom side.

```

```

void Editor::AlignBottoms () {
    Do(new AlignChange(drawing, drawingview, Bottom, Bottom));
}

// AlignTopSides aligns the rest of the Selections' top sides with the
// first Selection's top side.

void Editor::AlignTops () {
    Do(new AlignChange(drawing, drawingview, Top, Top));
}

// AlignVertCenters aligns the rest of the Selections' vertical
// centers with the first Selection's vertical center.

void Editor::AlignVertCenters () {
    Do(new AlignChange(drawing, drawingview, VertCenter, VertCenter));
}

// AlignHorizCenters aligns the rest of the Selections' horizontal
// centers with the first Selection's horizontal center.

void Editor::AlignHorizCenters () {
    Do(new AlignChange(drawing, drawingview, HorizCenter, HorizCenter));
}

// AlignCenters aligns the rest of the Selections' centers with the
// first Selection's center.

void Editor::AlignCenters () {
    Do(new AlignChange(drawing, drawingview, Center, Center));
}

// AlignLeftToRight aligns each Selection's left side with its
// predecessor's right side.

void Editor::AlignLeftToRight () {
    Do(new AlignChange(drawing, drawingview, Right, Left));
}

// AlignRightToLeft aligns each Selection's right side with its
// predecessor's left side.

void Editor::AlignRightToLeft () {
    Do(new AlignChange(drawing, drawingview, Left, Right));
}

// AlignBottomToTop aligns each Selection's bottom side with its
// predecessor's top side.

void Editor::AlignBottomToTop () {
    Do(new AlignChange(drawing, drawingview, Top, Bottom));
}

```

```

}

// AlignTopToBottom aligns each Selection's top side with its
// predecessor's bottom side.

void Editor::AlignTopToBottom () {
    Do(new AlignChange(drawing, drawingview, Bottom, Top));
}

// AlignToGrid aligns the Selections' lower left corners with the
// closest grid point.

void Editor::AlignToGrid () {
    Do(new AlignToGridChange(drawing, drawingview));
}

// Reduce reduces the drawing's magnification by a factor of two.

void Editor::Reduce () {
    drawingview->Reduce();
}

// Enlarge enlarges the drawing's magnification by a factor of two.

void Editor::Enlarge () {
    drawingview->Enlarge();
}

// NormalSize resets the drawing's magnification.

void Editor::NormalSize () {
    drawingview->NormalSize();
}

// ReduceToFit reduces the drawing's magnification enough to fit all
// of the drawing in the window.

void Editor::ReduceToFit () {
    drawingview->ReduceToFit();
}

// CenterPage scrolls the drawing so its center coincidences with the
// window's center.

void Editor::CenterPage () {
    drawingview->CenterPage();
}

// RedrawPage redraws the drawing without moving the view.

void Editor::RedrawPage () {
    drawingview->Draw();
}

```

```

}

// GriddingOnOff toggles the grid's constraint on or off.

void Editor::GriddingOnOff () {
    state->SetGridGravity(!state->GetGridGravity());
    state->UpdateViews();
}

// GridVisibleInvisible toggles the grid's visibility on or off.

void Editor::GridVisibleInvisible () {
    state->SetGridVisibility(!state->GetGridVisibility());
    drawingview->Draw();
}

// GridSpacing prompts the user for the new grid spacing in units of
// points.  If we didn't use points as units, the same grid spacing
// would not be portable to different displays.

void Editor::GridSpacing () {
    static char oldspacing[50];
    sprintf(oldspacing, "%lg", state->GetGridSpacing());
    char* spacing = nil;
    for (;;) {
        delete spacing;
        spacing = spacingdialog->Edit(oldspacing);
        if (spacing == nil) {
            break;
        }
        float s = 0.;
        if (sscanf(spacing, "%f", &s) == 1) {
            if (s > 0.) {
                state->SetGridSpacing(s);
                if (state->GetGridVisibility() == true) {
                    drawingview->Update();
                }
            }
            break;
        } else {
            spacingdialog->SetWarning("couldn't parse ", spacing);
        }
        delete spacing;
    }
}

// Orientation toggles the page between portrait and landscape
// orientations.

void Editor::Orientation () {
    state->ToggleOrientation();
    drawingview->Update();
}

```

```

}

// ShowVersion displays idraw's version level and author.

void Editor::ShowVersion () {
    versiondialog->Display();
}

// Construct the filename from the given prototype name

const char* Editor::MakeFilename(const char* name) {
    char* filename = new char [strlen(dir) + strlen(name) + GRAPH_EXT_LEN
+ 1];
    strcpy(filename,dir);
    strcat(filename,name);
    strcat(filename,GRAPH_EXT);
    return filename;
}

const char* Editor::MakeSpecFilename(const char* name) {
    char* filename = new char [strlen(dir) + strlen(name) +
SPEC_EXT_LEN + 1];
    strcpy(filename,dir);
    strcat(filename,name);
    strcat(filename,SPEC_PSDL_EXT);
    return filename;
}

// Do performs a change to the drawing and updates the drawing's
// modification status if it was unmodified.

void Editor::Do (ChangeNode* changenode) {
    switch (state->GetModifStatus()) {
        case Unmodified:
            history->Do(changenode);
            state->SetModifStatus(Modified);
            state->UpdateViews();
            break;
        default:
            history->Do(changenode);
            break;
    }
}

// InputVertices lets the user keep drawing a series of connected
// lines until the user presses a button other than the left button.
// It returns the vertices inputted by the user.

void Editor::InputVertices (Event& e, Coord*& xret, Coord*& yret, int&
nret) {
    const int INITIALSIZE = 100;
    static int sizebuffers = 0;

```

```

    static RubberLine** rubberlines = nil;
    static Coord* x = nil;
    static Coord* y = nil;
    if (INITIALSIZE > sizebuffers) {
sizebuffers = INITIALSIZE;
rubberlines = new RubberLine*[sizebuffers];
x = new Coord[sizebuffers];
y = new Coord[sizebuffers];
    }

    int n = 0;
    state->Constrain(e.x, e.y);
    rubberlines[0] = nil;
    x[0] = e.x;
    y[0] = e.y;
    ++n;

    while (e.button == LEFTMOUSE) {
rubberlines[n] = NewRubberLineOrAxis(e);
e.eventType = UpEvent;
drawingview->Manipulate(e, rubberlines[n], DownEvent, true, false);
Coord dummy;
rubberlines[n]->GetCurrent(dummy, dummy, e.x, e.y);
x[n] = e.x;
y[n] = e.y;
++n;

if (n == sizebuffers) {
    RubberLine** oldrubberlines = rubberlines;
    Coord* oldx = x;
    Coord* oldy = y;
    sizebuffers += INITIALSIZE/2;
    rubberlines = new RubberLine*[sizebuffers];
    x = new Coord[sizebuffers];
    y = new Coord[sizebuffers];
    bcopy(oldrubberlines, rubberlines, n * sizeof(RubberLine*));
    bcopy(oldx, x, n * sizeof(Coord));
    bcopy(oldy, y, n * sizeof(Coord));
    delete oldrubberlines;
    delete oldx;
    delete oldy;
}
}

    xret = x;
    yret = y;
    nret = n;
    for (int i = 1; i < n; i++) {
rubberlines[i]->Erase();
delete rubberlines[i];
    }
}

```

```

// NewRubberLineOrAxis creates and returns a new RubberLine or a new
// RubberAxis depending on whether the shift key's being depressed.

RubberLine* Editor::NewRubberLineOrAxis (Event& e) {
    return (!e.shift ?
        new RubberLine(nil, nil, e.x, e.y, e.x, e.y) :
        new RubberAxis(nil, nil, e.x, e.y, e.x, e.y));
}

// NewRubberEllipseOrCircle creates and returns a new RubberEllipse or
// a new RubberCircle depending on the shift key's state.

RubberEllipse* Editor::NewRubberEllipseOrCircle (Event& e) {
    return (!e.shift ?
        new RubberEllipse(nil, nil, e.x, e.y, e.x, e.y) :
        new RubberCircle(nil, nil, e.x, e.y, e.x, e.y));
}

// Skew comments/code ratio to work around cpp bug
// ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
// ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

// NewRubberRectOrSquare creates and returns a new RubberRect or a new
// RubberSquare depending on whether the shift key's being depressed.

RubberRect* Editor::NewRubberRectOrSquare (Event& e) {
    return (!e.shift ?
        new RubberRect(nil, nil, e.x, e.y, e.x, e.y) :
        new RubberSquare(nil, nil, e.x, e.y, e.x, e.y));
}

// OfferToSave returns true if it saves an unsaved drawing or the user
// refuses the offer or no changes need to be saved.

boolean Editor::OfferToSave () {
    boolean successful = false;
    if (state->GetModifStatus() == Modified) {
        char response = savecurdialog->Confirm();
        if (response == 'y') {
            Save();
            if (state->GetModifStatus() == Unmodified) {
                successful = true;
            }
        } else if (response == 'n') {
            successful = true;
        }
    } else {
        successful = true;
    }
    return successful;
}

```



```

// Reset redraws the view, clears the history, and resets state
// information about the drawing's name and its modification status.

void Editor::Reset (const char* filename, const char* prototype_name) {
    history->Clear();
    state->SetDrawingName(prototype_name);
    //    if (prototype_name != nil && !drawing->Writable(filename)) {
    //        state->SetModifStatus(ReadOnly);
    //    }
    //    else {
        state->SetModifStatus(Unmodified);
    //    }
    state->UpdateViews();
    drawingview->Update();
}

// ResetMessage changes the editor's message block

void Editor::ResetMessage(const char* msg) {
    state->SetMessage(msg);
    state->UpdateViews();
}

```

```

// file      idraw.h
// description: Class description of Idraw class (main class of editor).

// $Header: idraw.h,v 1.8 89/10/09 14:48:10 linton Exp $
// declares class Idraw.

/* Changse made to conform Idraw to CAPS graphic editor:
 * Add initial_prot variable to store initial prototype given and remove
 * initialfile variable because the editor is no longer file based.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 22, 1990
 */

#ifndef idraw_h
#define idraw_h

#include <InterViews/scene.h>

// Declare imported types.

class Drawing;
class DrawingView;
class Editor;
class ErrHandler;
class MapKey;
class State;

// An Idraw displays a drawing editor.

class Idraw : public MonoScene {
public:

    Idraw(int, char**);
    ~Idraw();

    void Run();

    void Handle(Event&);
    void Update();

protected:

    void ParseArgs(int, char**);
    void Init();

    const char* initial_prot; // stores name of initial prototype to
                               // use if any
    char* dir;                // stores name of directory of prototypes

    Drawing* drawing;         // performs operations on drawing

```

```
DrawingView* drawingview; // displays drawing
Editor* editor;           // handles drawing and editing operations
ErrorHandler* errhandler; // handles an X request error
MapKey* mapkey;           // maps characters to Interactors
State* state;              // stores current state info about drawing
Interactor* tools;        // displays drawing tools

};

#endif
```

```

// file      idraw.c
// description: Implementation of Idraw class.

// $Header: idraw.c,v 1.13 89/10/09 14:48:08 linton Exp $
// implements class Idraw.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Instead of basing all drawings on file names, make user use prototype
 * names and internally turn this into a file name by appending ".graph"
 * to it. This gives the impression that the graphic editor is prototype
 * name based instead of file based.
 * Instead of inputting file name, allow user to input prototype name and
 * directory where prototypes are stored.
 * Add glue to expand size of drawing area by 200 pixels.
 *
 * Changes made by: Mary Ann Cummings
 * Last change made: August 22, 1990
 */

#include "commands.h"
#include "drawing.h"
#include "drawingview.h"
#include "editor.h"
#include "errhandler.h"
#include "idraw.h"
#include "istring.h"
#include "mapkey.h"
#include "state.h"
#include "stateviews.h"
#include "tools.h"
#include <InterViews/Graphic/ppaint.h>
#include <InterViews/border.h>
#include <InterViews/box.h>
#include <InterViews/cursor.h>
#include <InterViews/event.h>
#include <InterViews/frame.h>
#include <InterViews/glue.h>
#include <InterViews/panner.h>
#include <InterViews/perspective.h>
#include <InterViews/sensor.h>
#include <InterViews/transformer.h>
#include <InterViews/tray.h>
#include <InterViews/Std/os/fs.h>
#include <sys/param.h>
#include <InterViews/Std/stdio.h>
#include <string.h>
#include <stdlib.h>

// Idraw parses its command line and initializes its members.

Idraw::Idraw (int argc, char** argv) {

```

```

        ParseArgs(argc, argv);
        InitPPaint();
        Init();
    }

    // Free storage allocated for members not in Idraw's scene.

    Idraw::~Idraw () {
    /*    delete drawing;
        delete dir;
        delete editor;
        delete errhandler;
        delete mapkey;
        delete state; */
    }

    // Run opens the initial file if one was given before starting to run.

    void Idraw::Run () {
        if (dir == nil) {

            //      use current directory  to search for and save drawing files

            const int bufsize = MAXPATHLEN + 1;
            static char buf[bufsize];
            getcwd(buf, bufsize);
            strcat(buf, "/");
            dir = buf;
        }

        // pass name of prototype directory to editor class

        editor->SetDirectory(dir);

        if (initial_prot != nil) {
            SetCursor(hourglass);
            editor->Open(initial_prot);
            SetCursor(defaultCursor);
        }

        Interactor::Run();
    }

    // Handle routes keystrokes to their associated interactors.

    void Idraw::Handle (Event& e) {
        switch (e.eventType) {
        case KeyEvent:
            if (e.len > 0) {
                Interactor* i = mapkey->LookUp(e.keystroke[0]);
                if (i != nil) {
                    i->Handle(e);
                }
            }
        }
    }

```

```

        }
    }
    break;
    default:
    break;
    }
}

// Update gets the picture's total tranformation matrix whenever it
// changes and stores it in State for creating new graphics.

void Idraw::Update () {
    Transformer t;
    drawing->GetPictureTT(t);
    state->SetGraphicT(t);
}

// ParseArgs stores the name of an initial prototype and name of prototype
// directory to use if any.

void Idraw::ParseArgs (int argc, char** argv) {
    dir = nil;
    initial_prot = nil;

    if (argc > 5) {
        fprintf(stderr, "too many arguments, usage: idraw [file]\n");
        const int PARSINGERROR = 1;
        exit(PARSINGERROR);
    }

    if (argc >= 3) {
//        set initial prototype

        if (strcmp(argv[1], "-p") == 0)
            initial_prot = argv[2];
        else {
//            set prototype directory

            if (strcmp(argv[1], "-d") == 0) {
                dir = new char [strlen(argv[2]) + 1];
                strcpy(dir, argv[2]);
                strcat(dir, "/");
            }
        }

        if (argc == 5) {
//            set initial prototype

```

```

        if (strcmp(argv[3], "-p") == 0)
            initial_prot = argv[4];
        else {

//      set prototype directory

            if (strcmp(argv[3], "-d") == 0) {
                dir = new char [strlen(argv[4]) + 1];
                strcpy(dir, argv[4]);
                strcat(dir, "/");
            }
        }
    }

// Init creates a sensor to catch keystrokes, creates members and
// initializes links between them, and composes them into a view with
// boxes, borders, glue, and frames.

void Idraw::Init () {
    input = new Sensor;
    input->Catch(KeyEvent);

    drawing      = new Drawing(8.5*inches, 11*inches, 0.05*inch);
    drawingview  = new DrawingView(drawing->GetPage());
    editor       = new Editor(this);
    errhandler   = new ErrHandler;
    mapkey       = new MapKey;
    state        = new State(this, drawing->GetPage());
    tools        = new Tools(editor, mapkey);

    drawingview->GetPerspective()->Attach(this);
    drawingview->SetSelectionList(drawing->GetSelectionList());
    drawingview->SetState(state);
    drawingview->SetTools(tools);
    editor->SetDrawing(drawing);
    editor->SetDrawingView(drawingview);
    editor->SetState(state);
    errhandler->SetEditor(editor);
    errhandler->Install();

    VBox* status = new VBox(
        new HBox(
            new ModifStatusView(state),
            new DrawingNameView(state),
            new GriddingView(state),
            new FontView(state),
            new MagnifView(state, drawingview)
        ),
        new HBorder
    );
};

```

```

VBox* msgblock = new VBox(
    new HBox(
        new HGlue(10,10),
        new MsgView(state)
    ),
    new HBorder
);
HBox* indic = new HBox(
    new BrushView(state),
    new VBorder,
    new PatternView(state)
);
HBox* cmds = new HBox(
    new Commands(editor, mapkey, state),

//    pad drawing area with 200 extra pixels

    new HGlue(200)
);

VBox* panel = new VBox(
    tools,
    new VGlue,
    new HBorder,
    new Panner(drawingview)
);
panel->Propagate(false);

HBorder* hborder = new HBorder;
VBorder* vborder = new VBorder;

Tray* t = new Tray;

t->HBox(t, status, t);
t->HBox(t, msgblock, t);
t->HBox(t, indic, vborder, cmds, t);
t->HBox(t, hborder, t);
t->HBox(t, panel, vborder, drawingview, t);

t->VBox(t, status, msgblock, indic, hborder, panel, t);
t->VBox(t, status, msgblock, vborder, t);
t->VBox(t, status, msgblock, cmds, hborder, drawingview, t);

Insert(new Frame(t, 1));
}

```



```

// file      istring.h
// description Defines needed string functions.

// $Header: istring.h,v 1.8 89/10/09 14:48:29 linton Exp $
// extends <string.h> to define strdup and strndup too.

/* No changes made to conform Idraw to CAPS graphic editor
*/

#ifndef istring_h
#define istring_h

#include <string.h>

// removes improper characters from a char string

char* RemoveBadChars(char*);
int LengthWithoutChars(char*, char*);

// create tempoary file name by retrieving environment variable
// TEMP and concantenating the filename onto it.

char* MakeTmpFileName(char*);

// strdup allocates and returns a duplicate of the given string.

inline char* strdup (const char* s) {
    char* dup = new char[strlen(s) + 1];
    strcpy(dup, s);
    return dup;
}

// strndup allocates and returns a duplicate of the first len
// characters of the given string.

inline char* strndup (const char* s, int len) {
    char* dup = new char[len + 1];
    strncpy(dup, s, len);
    dup[len] = '\0';
    return dup;
}

#endif

```

```

// file      istring.c
// description: Implements needed string functions.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add the implementation of string functions to help manipulate string
 * portion of PSDL.
 * This file was created specifically for graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 4, 1990
 */

#include "istring.h"
#include <InterViews/defs.h>

char* RemoveBadChars(char* string) {
    int new_len = LengthWithoutChars(string, " \n");
    int old_len = strlen(string);
    char* modified_string = new char[old_len + 1];
    if (new_len < old_len) {
        int m_ctr = 0;
        for (int i = 0; i < old_len; ++i) {
            if (string[i] != '\n' && string[i] != ' ') {
                modified_string[m_ctr++] = string[i];
            }
        }
        modified_string[m_ctr] = '\0';
    }
    else {
        strcpy(modified_string, string);
    }
    return modified_string;
}

int LengthWithoutChars(char* string, char* unwanted_chars) {
    int size = strlen(unwanted_chars);
    int string_size = strlen(string);
    int no_of_chars = 0;
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < string_size; ++j) {
            if (string[j] == unwanted_chars[i]) {
                ++no_of_chars;
            }
        }
    }
    return string_size - no_of_chars;
}

// make temporary filename by concatenating the environment variable

```

```

// TEMP with the given filename

char* MakeTmpFileName(char* filename) {
    char* tmp_dir = (char*) getenv("TEMP");
    char* result;

    if (tmp_dir != nil) {
        int tmp_len = strlen(tmp_dir);
        result = new char[tmp_len + strlen(filename) + 2];
        strcpy(result, tmp_dir);
        if(tmp_dir[tmp_len - 1] != '/') {
            strcat(result, "/");
        }
        strcat(result, filename);
    }
    else {
        result = filename;
    }
    return result;
}

```

```

// file          keystrokes.h
// description:  Defines keystrokes that map to tools and commands.

// $Header: keystrokes.h,v 1.12 89/10/09 14:48:30 linton Exp $
// defines all the key strokes that idraw handles.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Change RESHAPECHAR to MODIFYCHAR to be consistent with changing
 * RESHAPE to MODIFY as commands
 * Add ANNOTATECHAR, DECOMPOSECHAR, COMMENTCHAR, and LABELCHAR and
 * remove TEXTCHAR for the adding and removing of commands.
 * Change ANNOTATECHAR to SPECIFYCHAR to be consistent.
 * Add STREAMSCHAR and CONSTRAINTSCHAR because of adding those tools.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 3, 1990
 */

// You can select a PanelItem by typing one of these characters.

static const char SELECTCHAR    = 's';
static const char MOVECHAR      = 'm';
static const char SCALECHAR     = 'j';
static const char STRETCHCHAR   = 'i';
static const char ROTATECHAR    = 'k';
static const char MODIFYCHAR    = 'q';
static const char MAGNIFYCHAR   = 'z';
static const char SPECIFYCHAR   = 'A';
static const char STREAMSCHAR   = 'M';
static const char CONSTRAINTSCHAR = 'C';
static const char DECOMPOSECHAR = 'D';
static const char COMMENTCHAR   = 't';
static const char LABELCHAR     = 'T';
static const char METCHAR       = 'E';
static const char LATENCYCHAR   = 'L';
static const char LINECHAR      = 'l';
static const char MULTILINECHAR = 'w';
static const char BSPLINECHAR   = 'h';
static const char ELLIPSECHAR   = 'o';
static const char RECTCHAR      = 'r';
static const char POLYGONCHAR   = 'p';
static const char CLOSEDBSPLINECHAR = 'y';

// You can execute a PullDownMenuCommand by typing one of these
// characters.

static const char NEWCHAR       = '\016'; // ^N
static const char REVERTCHAR    = '\022'; // ^R
static const char OPENCHAR      = '\017'; // ^O
static const char SAVECHAR      = '\023'; // ^S
static const char SAVEASCHAR    = '\001'; // ^A

```

```

static const char PRINTCHAR    = '\020'; // ^P
static const char QUITCHAR     = '\021'; // ^Q

static const char UNDOCHAR     = 'U';
static const char REDOCHAR     = 'R';
static const char CUTCHAR      = 'x';
static const char COPYCHAR     = 'c';
static const char PASTECHAR    = 'v';
static const char DUPLICATECHAR = 'd';
static const char DELETECHAR   = '\004'; // ^D
static const char SELECTALLCHAR = 'a';
static const char FLIPHORIZONTALCHAR = '_';
static const char FLIPVERTICALCHAR = '|';
static const char _90CLOCKWISECHAR = ']';
static const char _90COUNTERCWCHAR = '[';
static const char PRECISEMOVECHAR = 'M';
static const char PRECISESCALECHAR = 'J';
static const char PRECISEROTATECHAR = 'K';

static const char GROUPCHAR    = 'g';
static const char UNGROUPCHAR  = 'u';
static const char BRINGTOFRONTCHAR = 'f';
static const char SENDTOBACKCHAR = 'b';
static const char NUMBEROFGRAHICSCHAR = '#';

static const char ALIGNLEFTSIDESCHAR = '1';
static const char ALIGNRIGHTSIDESCHAR = '2';
static const char ALIGNBOTTOMSCHAR = '3';
static const char ALIGNTOPSCHAR = '4';
static const char ALIGNVERTCENTERSCHAR = '5';
static const char ALIGNHORIZCENTERSCHAR = '6';
static const char ALIGNCENTERSCHAR = '7';
static const char ALIGNLEFTTORIGHTCHAR = '8';
static const char ALIGNRIGHTTOLEFTCHAR = '9';
static const char ALIGNBOTTOMTOTOPCHAR = '0';
static const char ALIGNTOPTOBOTTOMCHAR = '-';
static const char ALIGNTOGRIDCHAR = '.';

static const char REDUCECHAR    = 'i';
static const char ENLARGECHAR   = 'e';
static const char NORMALSIZECHAR = 'n';
static const char REDUCETO FITCHAR = '=';
static const char CENTERPAGECHAR = '/';
static const char REDRAWPAGECHAR = '\014'; // ^L
static const char GRIDDINGONOFFCHAR = ',';
static const char GRIDVISIBLEINVISIBLECHAR = '?';
static const char GRIDSPACINGCHAR = 'S';
static const char ORIENTATIONCHAR = '+';
static const char SHOWVERSIONCHAR = '$';

```

```

// file          main.c
// description    Graphic editor driver.

// $Header: main.c,v 1.11 89/10/09 14:48:57 linton Exp $
// runs idraw.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Use only one brush type (directed line with arrowhead on right hand end
 * of line) and comment out all other brush options
 * Add options for pattern for arrowhead
 * Comment out all but three pattern choices since the others are not
needed
 * for data flow diagrams
 * Change initial brush pattern to be the only one available
 * Change initial pattern to be white
 * Add geometry option to make graphic editor appear in lower left hand
corner
 * of screen
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 29, 1990
 */

#include "idraw.h"
#include <InterViews/world.h>

// Predefine default properties for the window size, paint menus, and
// history.

static PropertyData properties[] = {
    { "**font1",    "**-courier-medium-r*-80-*    Courier 8" },
    { "**font2",    "**-courier-medium-r*-100-*    Courier 10" },
    { "**font3",    "**-courier-bold-r*-120-*    Courier-Bold 12" },
    { "**font4",    "**-helvetica-medium-r*-120-*  Helvetica 12" },
    { "**font5",    "**-helvetica-medium-r*-140-*  Helvetica 14" },
    { "**font6",    "**-helvetica-bold-r*-140-*    Helvetica-Bold 14" },
    { "**font7",    "**-helvetica-medium-o*-140-*  Helvetica-Oblique 14" },
    { "**font8",    "**-times-medium-r*-120-*    Times-Roman 12" },
    { "**font9",    "**-times-medium-r*-140-*    Times-Roman 14" },
    { "**font10",   "**-times-bold-r*-140-*    Times-Bold 14" },
    { "**font11",   "**-times-medium-i*-140-*    Times-Italic 14" },
    { "**brush1",   "ffff 1 0 1" },

    // These brush types are not needed for a data flow diagram

    /* ***** Start of Commented Out Code *****
    { "**brush1",   "none" },
    { "**brush2",   "ffff 1 0 0" },      removed all brush types except for
    { "**brush3",   "ffff 1 1 0" },      the type that has as its first end
    { "**brush4",   "ffff 1 0 1" },      endpoint the end without the arrow
    { "**brush5",   "ffff 1 1 1" },      head and it's second or last end
    */

```

```

{ **brush6", "3333 1 0 0" },    point is the end with the arrowhead
{ **brush7", "3333 2 0 0" },
{ **brush8", "ffff 2 0 0" },
***** End of Comments Out Code ***** */

```

```

{ **pattern1", "none" },
{ **pattern2", "0.0" },
{ **pattern3", "1.0" },
{ **arrowpattern1", "none" },
{ **arrowpattern2", "0.0" },
{ **arrowpattern3", "1.0" },

```

// These patterns are not needed for a data flow diagram

/* ***** Start of Commented Out Code *****

```

{ **pattern4", "0.75" },
{ **pattern5", "0.5" },
{ **pattern6", "0.25" },
{ **pattern7", "1248" },
{ **pattern8", "8421" },
{ **pattern9", "f000" },
{ **pattern10", "8888" },
{ **pattern11", "f888" },
{ **pattern12", "8525" },
{ **pattern13", "cc33" },
{ **pattern14", "7bed" },
***** End of Commented Out Code ***** */

```

```

{ **fgcolor1", "Black" },
{ **fgcolor2", "Brown 42240 10752 10752" },
{ **fgcolor3", "Red" },
{ **fgcolor4", "Orange" },
{ **fgcolor5", "Yellow" },
{ **fgcolor6", "Green" },
{ **fgcolor7", "Blue" },
{ **fgcolor8", "Indigo 48896 0 65280" },
{ **fgcolor9", "Violet 20224 12032 20224" },
{ **fgcolor10", "White" },
{ **fgcolor11", "LtGray 50000 50000 50000" },
{ **fgcolor12", "DkGray 33000 33000 33000" },
{ **bgcolor1", "Black" },
{ **bgcolor2", "Brown 42240 10752 10752" },
{ **bgcolor3", "Red" },
{ **bgcolor4", "Orange" },
{ **bgcolor5", "Yellow" },
{ **bgcolor6", "Green" },
{ **bgcolor7", "Blue" },
{ **bgcolor8", "Indigo 48896 0 65280" },
{ **bgcolor9", "Violet 20224 12032 20224" },
{ **bgcolor10", "White" },
{ **bgcolor11", "LtGray 50000 50000 50000" },
{ **bgcolor12", "DkGray 33000 33000 33000" },

```

```

    { "**initialfont", "2" },
    { "**initialbrush", "1" },
    { "**initialpattern", "3" },
    { "**initialarrowpattern", "2" },
    { "**initialfgcolor", "1" },
    { "**initialbgcolor", "10" },
    { "**history", "20" },
    { "**reverseVideo", "off" },
    { "**small", "true" },
    { "**geometry", "+0-0"},
    { nil }
};

// Define window size options.

static OptionDesc options[] = {
    { "-l", "**small", OptionValueImplicit, "false" },
    { "-s", "**small", OptionValueImplicit, "true" },
    { nil }
};

// main creates a connection to the display server, creates idraw, and
// opens idraw's window. After idraw stops running, main closes
// idraw's window, deletes everything it created, and returns success.

int main (int argc, char** argv) {
    World* world = new World("Idraw", properties, options, argc, argv);
    Idraw* idraw = new Idraw(argc, argv);

    world->InsertApplication(idraw);
    idraw->Run();
    world->Remove(idraw);

    delete idraw;
    delete world;

    const int SUCCESS = 0;
    return SUCCESS;
}

```



```

// file          opsellist.h
// description:   Class description of OperatorSelList.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add definition of list of operator selections class and class of nodes
 * to be used in the list
 * This header file was created specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 29, 1990
 */

#ifndef opsellist_h
#define opsellist_h

#include "list.h"

class BSplineSelection;
class EllipseSelection;
// class LineSelection;
class OperatorSelection;
class Selection;
class TextSelection;

class OperatorSelNode : public BaseNode {
public:
    OperatorSelNode(OperatorSelection* os) { opsel = os; }
    boolean SameValueAs(void* p) { return opsel == p; }
    OperatorSelection* GetSelection() { return opsel; }

protected:
    OperatorSelection* opsel;    // points to an operator selection
};

class OperatorSelList : public BaseList {
public:
    OperatorSelNode* First();
    OperatorSelNode* Last();
    OperatorSelNode* Prev();
    OperatorSelNode* Next();
    OperatorSelNode* GetCur();
    void SetCur(EllipseSelection*);
    void SetCurContaining(Coord, Coord);
    void SetCurWithLabel(TextSelection*);
    OperatorSelNode* Index(int);
    void SearchOperators(TextSelection*, EllipseSelection*);
// void SearchOperatorsForLine(TextSelection*, LineSelection*);
    char* SearchOperatorsForSpline(TextSelection*, BSplineSelection*);
    void Replace(Selection*, Selection*);
    void Remove(Selection*);
    void AddMETToOperator(TextSelection*, EllipseSelection*);

```

```

        void AddLatencyToDataFlow(TextSelection*, BSplineSelection*);
        Selection* FindLabel(Selection*);
        BSplineSelection* GetFlow(TextSelection*);
        TextSelection* GetSelfLoopLabel(BSplineSelection*);
        TextSelection* GetDFLabel(BSplineSelection*);
        TextSelection* GetDFLatency(BSplineSelection*);
        TextSelection* GetOperatorLabel(EllipseSelection*);
        TextSelection* GetOperatorMET(EllipseSelection*);
    protected:
        void ReplaceOperator(EllipseSelection*, EllipseSelection*);
        void ReplaceText(TextSelection*, TextSelection*);
    // void ReplaceDFLine(LineSelection*, LineSelection*);
        void ReplaceDFSpline(BSplineSelection*, BSplineSelection*);
        void ReplaceSelfLoop(BSplineSelection*, BSplineSelection*);
        void RemoveOperator(EllipseSelection*);
    // void RemoveDFLine(LineSelection*);
        void RemoveDFSpline(BSplineSelection*);
        void RemoveSelfLoop(BSplineSelection*);
};

inline OperatorSelNode* OperatorSelList::First() {
    return (OperatorSelNode*) BaseList::First();
}

inline OperatorSelNode* OperatorSelList::Last() {
    return (OperatorSelNode*) BaseList::Last();
}

inline OperatorSelNode* OperatorSelList::Prev() {
    return (OperatorSelNode*) BaseList::Prev();
}

inline OperatorSelNode* OperatorSelList::Next() {
    return (OperatorSelNode*) BaseList::Next();
}

inline OperatorSelNode* OperatorSelList::GetCur() {
    return (OperatorSelNode*) BaseList::GetCur();
}

inline OperatorSelNode* OperatorSelList::Index(int index) {
    return (OperatorSelNode*) BaseList::Index(index);
}

#endif

```

```

// file      opsellist.c
// description: Implementation of OperatorSelList class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add implementation of list of operator selections class.
 * This file was created specifically for the graphic editor.
 *
 * Changes made by:  Mary Ann Cummings
 * Last change made: August 29, 1990
 */

#include "dfdclasses.h"
#include "dfdsplinelist.h"
#include "opsellist.h"
#include "selection.h"
#include "sldfdline.h"
#include "sldfdspline.h"
#include "sl ellipses.h"
#include "sloperator.h"
#include "slsplines.h"
#include "sltext.h"
#include <InterViews/Graphic/base.h>
#include <InterViews/Std/string.h>

// Set current node of list to be the one that contains the given ellipse
// selection (operator)

void OperatorSelList::SetCur(EllipseSelection* es) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetEllipseSelection() == es) {
            return;
        }
    }
    return;
}

// Set current node of list to be the one whose ellipse selection contains
// the given point

void OperatorSelList::SetCurContaining(Coord x, Coord y) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetEllipseSelection()->Contains(
                                                    new PointObj(x,y)))
            return;
    }
    return;
}

// Set current node of list to be the one whose txtsel is the given
// TextSelection

void OperatorSelList::SetCurWithLabel(TextSelection* label) {

```

```

        for (First(); !AtEnd(); Next()) {
            if (GetCur()->GetSelection()->GetTextSelection() == label) {
                return;
            }
        }
        return;
    }

// Search list for given ellipse selection (operator) and if found, place
given
// text (label) in that node

void OperatorSelList::SearchOperators(TextSelection* ts,
                                      EllipseSelection* es) {
    SetCur(es);
    if (!AtEnd())
        GetCur()->GetSelection()->SetTextSelection(ts);
}

// Lines no longer used, only use splines

/* ***** Start of Commented Out Code *****

// Search list for given line in each of the operator's list of lines and
if
// found, place given text (label) in that node in the list of lines.
// Because we have to find both operators in order to update the PSDL
// properly, the function of finding the line in the list has been split
// into 2 functions. The first function adds the label, the second func-
tion
// just updates the PSDL.

void OperatorSelList::SearchOperatorsForLine(TextSelection* ts,
                                              LineSelection* ls) {
    char* old_string = nil;
    int len;
    const char* copy_string;
    OperatorSelection* first_op;
    char* new_string;
    copy_string = ts->GetOriginal(len);
    new_string = new char[len+1];
    strncpy(new_string, copy_string, len);
    new_string[len] = '\0';
    for (First(); !AtEnd(); Next()) {
        old_string = GetCur()->GetSelection()->FindLineInList(ts, ls);
        if (old_string != nil) {
            first_op = GetCur()->GetSelection();
            break;
        }
    }

    for (First(); !AtEnd(); Next()) {

```

```

        if (GetCur()->GetSelection() != first_op) {
            boolean found_2 = GetCur()->GetSelection()
                               ->FindLineInSecondList(old_string, new_string, ls);
            if (found_2)
                break;
        }
    }
}

***** End of Commented Out Code ***** */

// Search list for given spline in each of operator's list of splines and
// if found, place given text (label) in that node in the list of splines.
// Because we have to find both operators in order to update the PSDL
// properly, the function of finding the line in the list has been split
// into finding the first line to update the label and finding the line in
// the second list to change the PSDL.

char* OperatorSelList::SearchOperatorsForSpline(TextSelection* ts,
                                                BSplineSelection* ss) {
    char* old_string = nil;
    const char* copy_string;
    char* new_string;
    int len;
    copy_string = ts->GetOriginal(len);
    new_string = new char[len+1];
    strncpy(new_string, copy_string, len);
    new_string[len] = '\0';
    OperatorSelection* first_op;
    for (First(); !AtEnd(); Next()) {
        old_string = GetCur()->GetSelection()->FindSplineInList(ts, ss);
        if (old_string != nil) {
            first_op = GetCur()->GetSelection();
            break;
        }
    }

    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection() != first_op) {
            boolean found_2 = GetCur()->GetSelection()
                               ->FindSplineInSecondList(old_string, new_string, ss);
            if (found_2) {
                return old_string;
            }
        }
    }
    return nil;
}

// Search list for given ellipse selection (operator) and if found,
// replace it with the new ellipse selection (operator)

void OperatorSelList::ReplaceOperator(EllipseSelection* replacee,

```

```

                                EllipseSelection* replacer) {
    SetCur(replacee);
    if (!AtEnd()) {
        GetCur()->GetSelection()->SetEllipseSelection(replacer);
    }
}

// Search list for given text (label) and if found, replace it with new
// text (label).  If the label is not attached to an operator, each of
// of the lists associated with the operator must be searched because
// it might be a label on one of the data flows or self loops

void OperatorSelList::ReplaceText(TextSelection* replacee,
                                TextSelection* replacer) {
    boolean found = false;
    ClassId cid = replacee->GetClassId();
    char* old_string;
    switch (cid) {
    case LABEL_OP:
        for (First(); !AtEnd() && !found; Next()) {
            if (GetCur()->GetSelection()->GetTextSelection() == replacee) {
                found = true;
                GetCur()->GetSelection()->SetTextSelection(replacer);
            }
        }
        break;
    case MET_OP:
        for (First(); !AtEnd() && !found; Next()) {
            if (GetCur()->GetSelection()->GetMETSelection() == replacee) {
                found = true;
                GetCur()->GetSelection()->SetMETSelection(replacer);
            }
        }
        break;
    case LAT_DF:
    case LABEL_SL:
        for (First(); !AtEnd() && !found; Next()) {
            old_string = GetCur()->GetSelection()->
                FoundTextInLists(replacee, replacer);
            if (old_string != nil) {
                found = true;
            }
        }
        break;
    case LABEL_DF:
        for (First(); !AtEnd(); Next()) {
            if (!found) {
                old_string = GetCur()->GetSelection()->
                    FoundTextInLists(replacee, replacer);
                if (old_string != nil) {
                    found = true;
                }
            }
        }
    }
}

```

```

        }
        else {
            GetCur()->GetSelection()
                ->FoundSecondTextInLists(old_string, replacer);
        }
    }
}

// Determine the type (class id) of the given selection and call proper
// function to replace the selection with the new selection in the proper
// list

void OperatorSelList::Replace(Selection* replacee, Selection* replacer) {
    ClassId cid = replacee->GetClassId();
    if (cid == OPERATOR) {
        ReplaceOperator((EllipseSelection*) replacee,
            (EllipseSelection*) replacer);
    }
    else {
        if (cid == LABEL_OP || cid == LABEL_DF || cid == LABEL_SL ||
            cid == MET_OP || cid == LAT_DF) {
            ReplaceText((TextSelection*) replacee,
                (TextSelection*) replacer);
        }
        /* ***** Start of Commented Out Code *****
        else {
            if (cid == DATAFLOW_LINE) {
                ReplaceDFLine((LineSelection*) replacee,
                    (LineSelection*) replacer);
            }
            ***** End of Commented Out Code ***** */
        else {
            if (cid == DATAFLOW_SPLINE) {
                ReplaceDFSpline((BSplineSelection*) replacee,
                    (BSplineSelection*) replacer);
            }
            else {
                if (cid == SELFLOOP) {
                    ReplaceSelfLoop((BSplineSelection*) replacee,
                        (BSplineSelection*) replacer);
                }
            }
        }
    }
}

// }
}

}

// No longer using lines, use splines instead

/* ***** Start of Commented Out Code *****

```

```

// Search each list of lines associated with each operator in list to re-
place
// given line with new line

void OperatorSelList::ReplaceDFLine(LineSelection* replacee,
                                   LineSelection* replacer) {
    boolean found = false;
    for (First(); !AtEnd() && !found; Next()) {
        found = GetCur()->GetSelection()->FoundLineInLists(replacee, re-
placer);
    }
}

***** End of Commented Out Code ***** */

// Search each list of splines associated with each operator in list to
// replace given spline with new spline

void OperatorSelList::ReplaceDFSpline(BSplineSelection* replacee,
                                       BSplineSelection* replacer) {
    boolean found = false;
    for (First(); !AtEnd() && !found; Next()) {
        found = GetCur()->GetSelection()->
FoundSplineInLists(replacee, replacer);
    }
}

// Search list of self loops associated with each operator in list to
// replace given self loop with new self loop

void OperatorSelList::ReplaceSelfLoop(BSplineSelection* replacee,
                                       BSplineSelection* replacer) {
    boolean found = false;
    for (First(); !AtEnd() && !found; Next()) {
        found = GetCur()->GetSelection()->
FoundSelfLoopInList(replacee, replacer);
    }
}

// Remove the given selection from the operator selection list

void OperatorSelList::Remove(Selection* s) {
    ClassId cid = s->GetClassId();
    if (cid == OPERATOR) {
        RemoveOperator((EllipseSelection*) s);
    }
/* ***** Start of Commented Out Code *****
    else {
        if (cid == DATAFLOW_LINE) {
            RemoveDFLine((LineSelection*) s);
        }
        ***** End of Commented Out Code ***** */
    else {

```



```

        if (cid == DATAFLOW_SPLINE) {
            RemoveDFSpline((BSplineSelection*) s);
        }
        else {
            if (cid == SELFLOOP) {
                RemoveSelfLoop((BSplineSelection*) s);
            }
            else {
                if (cid == LABEL_OP || cid == LABEL_DF || cid == LABEL_SL ||
                    cid == MET_OP || cid == LAT_DF) {
                    //      a node does not need to be deleted, only the label of the node
                    //      needs to be deleted
                    TextSelection* ts = nil;
                    ReplaceText((TextSelection*) s, ts);
                }
            }
        }
    }
}

// Remove an operator from the list of operator selections

void OperatorSelList::RemoveOperator(EllipseSelection* es) {
    SetCur(es);
    if (!AtEnd()) {
        OperatorSelection* os = GetCur()->GetSelection();
        DeleteCur();
        delete os;
    }
}

// Lines no longer used, use splines instead

/* ***** Start of Commented Out Code *****

// remove a data flow line from the operator selection list. This
// involves removing it from both operator selection nodes if it is
// an output of one operator and an input of another operator. Otherwise,
// the node will only be deleted once. Return the line's label so that it
// may be removed also.

void OperatorSelList::RemoveDFLine(LineSelection* ls) {
    boolean found_1 = false, found_2 = false;
    DFDLineSelection* DFDL;

    // search each operator

    for (First(); !AtEnd(); Next()) {
        DFDLineSelList* ill = GetCur()->GetSelection()->GetInputDFLineL-
ist();

```

```

//      search the operator's list of input data flow lines

for (ill->First(); !ill->AtEnd(); ill->Next()) {
    if (ill->GetCur()->GetSelection()->GetLineSelection() == ls) {

//          the line was found in the input list

                if (!found_1) {
                    found_1 = true;
                    DFDL = ill->GetCur()->GetSelection();
                }
                else {
                    found_2 = true;
                }
                ill->DeleteCur();

//          an assumption was made that it will not be in the input
//          list more than once

                break;
            }
        }

//      if line has not been found in two lists, search output list

if (!found_2) {
    DFDLineSellList* oll = GetCur()->GetSelection()->
                                GetOutputDFLineList();
    for (oll->First(); !oll->AtEnd(); oll->Next()) {
        if (oll->GetCur()->GetSelection()->GetLineSelection() ==
ls) {

//            the line was found in the output list

                    if (!found_1) {
                        found_1 = true;
                        DFDL = oll->GetCur()->GetSelection();
                    }
                    else {
                        found_2 = true;
                    }
                    oll->DeleteCur();

//            an assumption was made that the line will not be in
//            the output list more than once

                    break;
                }
            }
        }
    }
    if (found_2) {

```

```

//      the line was found in two lists so we do not need to search
//      the operator list any longer

//      remove the dfd line selection that the node was pointing to

        delete DFDL;
        break;
    }
}
if (found_1 && (!found_2)) {
//      remove the dfd line selection that the node was pointing to

        delete DFDL;
    }
}
***** End of Commented Out Code ***** */

// remove a data flow spline from the operator selection list. This
// involves removing it from both operator selectio nodes if it is
// an output of one operator and an input of another operator. Otherwise,
// the node will only be deleted once. Return the spline's label so that
// it may be removed also.

void OperatorSelList::RemoveDFSpline(BSplineSelection* ss) {
    boolean found_1 = false, found_2 = false;
    DFDSplineSelection* DFDS;

//    search each operator

    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* isl = GetCur()->GetSelection()->
            GetInputDFSplineList();

//        search the operator's list of input data flow splines

        for (isl->First(); !isl->AtEnd(); isl->Next()) {
            if (isl->GetCur()->GetSelection()->GetSplineSelection() == ss) {

//                the line was found in the input list

                if (!found_1) {
                    found_1 = true;
                    DFDS = isl->GetCur()->GetSelection();
                }
                else {
                    found_2 = true;
                }
                TextSelection* in_label_ts = isl->GetCur()->GetSelection()
                    ->GetTextSelection();
            }
        }
    }
}

```

```

        GetCur()->GetSelection()->RemoveInputFromPSDL(in_label_ts);
        isl->DeleteCur();

//      an assumption was made that it will not be in the input
//      list more than once

        break;
    }
}

//  if the spline has not been found in two lists, search the output
list

    if (!found_2) {
        DFDSplineSelList* osl = GetCur()->GetSelection()->
                                GetOutputDFSplineList();
        for (osl->First(); !osl->AtEnd(); osl->Next()) {
            if (osl->GetCur()->GetSelection()->
                GetSplineSelection() == ss) {

//          the spline was found in the output list

                if (!found_1) {
                    found_1 = true;
                    DFDS = osl->GetCur()->GetSelection();
                }
                else {
                    found_2 = true;
                }
                TextSelection* out_label_ts = osl->GetCur()->GetSe-
lection()
                                ->GetTextSelection();
                GetCur()->GetSelection()->
                    RemoveOutputFromPSDL(out_label_ts);
                osl->DeleteCur();

//          an assumption was made that the spline will not be in
//          the output list more than once

                break;
            }
        }
    }
    if (found_2) {

//      the spline was found in two lists so we do not need to search the
//      operator list any longer

//      remove the dfd spline selection that the node was pointing to
        delete DFDS;
    }
}

```

```

        break;
    }
}
if (found_1 && (!found_2)) {
//    remove the dfd spline selection that the node was pointing to
    delete DFDS;
}
}

// remove a self loop from the operator selection list. Return the self
// loop's label so that it may be removed too.

void OperatorSelList::RemoveSelfLoop(BSplineSelection* ss) {
    boolean found = false;
    DFDSplineSelection* DFDS;

//    search the list of operators

    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* sll = GetCur()->GetSelection()->GetSelfLoop-
List();

//        search the operator's list of self loops

        for (sll->First(); !sll->AtEnd(); sll->Next()) {
            if (sll->GetCur()->GetSelection()->GetSplineSelection() == ss) {

//                the self loop was found

                TextSelection* label_ts = sll->GetCur()->GetSelection()
                    ->GetTextSelection();
                GetCur()->GetSelection()->RemoveStateFromPSDL(label_ts);
                found = true;
                DFDS = sll->GetCur()->GetSelection();
                sll->DeleteCur();

//                delete the dfd spline selection that the node is pointing to

                delete DFDS;

//                stop searching the list of self loops

                break;
            }
        }
        if (found)

//            stop searching the list of operators

            break;
    }
}

```

```

    }
}

void OperatorSelList::AddMETToOperator(TextSelection* met_ts,
                                       EllipseSelection* op) {
    SetCur(op);
    if (!AtEnd()) {
        GetCur()->GetSelection()->SetMETSelection(met_ts);
    }
}

// Search list for given spline in each of operator's list of splines and
// if found, place given text (latency) in that node in the list of
// splines.

void OperatorSelList::AddLatencyToDataFlow(TextSelection* ts,
                                           BSplineSelection* ss) {
    boolean found = false;
    for (First(); !AtEnd(); Next()) {
        found = GetCur()->GetSelection()->AddLatencyToSplineInList(ts,
ss);
        if (found)
            return;
    }
}

// Given a selection, FindLabel attempts to find the label associated
// with it. If the label is nil or the the selection is not in the
// list, the given selection is returned. This is used by Editor,
// to select the label of the chosen object. Nil could not be returned,
// because this object will used by the chosen tool

Selection* OperatorSelList::FindLabel(Selection* sel) {
    TextSelection* ts;
    ClassId cid = sel->GetClassId();
    switch (cid) {
    case OPERATOR:
        for (First(); !AtEnd(); Next()) {
            if (GetCur()->GetSelection()->GetEllipseSelection() == sel) {
                ts = GetCur()->GetSelection()->GetTextSelection();
                if (ts != nil) {
                    return ts;
                }
            }
            else {
                return sel;
            }
        }
        return sel;
        break;
    case DATAFLOW_SPLINE:
        for (First(); !AtEnd(); Next()) {
            DFDSplineSelList* il = GetCur()->GetSelection()

```

```

                                ->GetInputDFSplineList();
for (il->First(); !il->AtEnd(); il->Next()) {
    if (il->GetCur()->GetSelection()->GetSplineSelection()==sel)
    {
        ts = il->GetCur()->GetSelection()->GetTextSelection();
        if (ts != nil) {
            return ts;
        }
        else {
            return sel;
        }
    }
}
DFDSplineSelList* ol = GetCur()->GetSelection()
                                ->GetOutputDFSplineList();
for (ol->First(); !ol->AtEnd(); ol->Next()) {
    if (ol->GetCur()->GetSelection()->GetSplineSelection()==sel)
    {
        ts = ol->GetCur()->GetSelection()->GetTextSelection();
        if (ts != nil) {
            return ts;
        }
        else {
            return sel;
        }
    }
}
return sel;
break;
case SELFLOOP:
    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* sll = GetCur()->GetSelection()
                                ->GetSelfLoopList();
        for (sll->First(); !sll->AtEnd(); sll->Next()) {
            if (sll->GetCur()->GetSelection()->GetSplineSelection()==
sel) {
                ts = sll->GetCur()->GetSelection()->GetTextSelection();
                if (ts != nil) {
                    return ts;
                }
                else {
                    return sel;
                }
            }
        }
    }
return sel;
break;
default:
    return sel;
break;

```

```

    }
}

// Find flow for given data flow's label

BSplineSelection* OperatorSelList::GetFlow(TextSelection* ts) {
    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* il = GetCur()->GetSelection()->GetInputDFS-
plineList();
        for (il->First(); !il->AtEnd(); il->Next()) {
            if (il->GetCur()->GetSelection()->GetTextSelection() == ts) {
                return il->GetCur()->GetSelection()->GetSplineSelection();
            }
        }
        DFDSplineSelList* ol =
            GetCur()->GetSelection()->GetOutputDFSplineList();
        for (ol->First(); !ol->AtEnd(); ol->Next()) {
            if (ol->GetCur()->GetSelection()->GetTextSelection() == ts) {
                return ol->GetCur()->GetSelection()->GetSplineSelection();
            }
        }
    }
    return nil;
}

// returns label of given self loop

TextSelection* OperatorSelList::GetSelfLoopLabel(BSplineSelection* sl) {
    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* sll = GetCur()->GetSelection()->GetSelfLoop-
List();
        for (sll->First(); !sll->AtEnd(); sll->Next()) {
            if (sll->GetCur()->GetSelection()->GetSplineSelection() == sl) {
                return sll->GetCur()->GetSelection()->GetTextSelection();
            }
        }
    }
    return nil;
}

// returns label of given data flow

TextSelection* OperatorSelList::GetDFLabel(BSplineSelection* df) {
    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* il =
            GetCur()->GetSelection()->GetInputDFSplineList();
        for (il->First(); !il->AtEnd(); il->Next()) {
            if (il->GetCur()->GetSelection()->GetSplineSelection() == df) {
                return il->GetCur()->GetSelection()->GetTextSelection();
            }
        }
        DFDSplineSelList* ol =
            GetCur()->GetSelection()->GetOutputDFSplineList();
    }
}

```



```

        for (ol->First(); !ol->AtEnd(); ol->Next()) {
            if (ol->GetCur()->GetSelection()->GetSplineSelection() == df) {
                return ol->GetCur()->GetSelection()->GetTextSelection();
            }
        }
    }
    return nil;
}

// returns latency of given data flow

TextSelection* OperatorSelList::GetDFLatency(BSplineSelection* df) {
    for (First(); !AtEnd(); Next()) {
        DFDSplineSelList* il =
            GetCur()->GetSelection()->GetInputDFSplineList();
        for (il->First(); !il->AtEnd(); il->Next()) {
            if (il->GetCur()->GetSelection()->GetSplineSelection() == df) {
                return il->GetCur()->GetSelection()->GetLatencySelection();
            }
        }
        DFDSplineSelList* ol =
            GetCur()->GetSelection()->GetOutputDFSplineList();
        for (ol->First(); !ol->AtEnd(); ol->Next()) {
            if (ol->GetCur()->GetSelection()->GetSplineSelection() == df) {
                return ol->GetCur()->GetSelection()->GetLatencySelection();
            }
        }
    }
    return nil;
}

// returns label of given operator

TextSelection* OperatorSelList::GetOperatorLabel(EllipseSelection* op) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetEllipseSelection() == op) {
            return GetCur()->GetSelection()->GetTextSelection();
        }
    }
    return nil;
}

// returns maximum execution time of given operator

TextSelection* OperatorSelList::GetOperatorMET(EllipseSelection* op) {
    for (First(); !AtEnd(); Next()) {
        if (GetCur()->GetSelection()->GetEllipseSelection() == op) {
            return GetCur()->GetSelection()->GetMETSelection();
        }
    }
    return nil;
}

```

```

// file      sldfdspline.h
// description: Class description of DFDSplineSelection class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add definition of dfd spline selection class.
 * This header file was created specifically for graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 30, 1990
 */

#ifndef sldfdspline_h
#define sldfdspline_h

class BSplineSelection;
class TextSelection;

class DFDSplineSelection {
public:
    DFDSplineSelection(BSplineSelection*);
    ~DFDSplineSelection();
    void SetSplineSelection(BSplineSelection*);
    BSplineSelection* GetSplineSelection();
    void SetTextSelection(TextSelection*);
    TextSelection* GetTextSelection();
    void SetLatencySelection(TextSelection*);
    TextSelection* GetLatencySelection();

protected:
    BSplineSelection* splsel;    // curve data flow line or self loop
    TextSelection* txtsel;      // label of curved data flow line or
                                // self loop
    TextSelection* latsel;      // latency of data flow
};

#endif

```

```

// file          sldfdspline.c
// description:  Implementation of DFDSplineSelection class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add implementation of dfd spline selection. This represents
 * curved data flows or self loops and their labels.
 * This file was created specifically for the graphic editor
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 30, 1990
 */

#include "sldfdspline.h"
#include "splsplines.h"
#include "slttext.h"
#include <InterViews/Graphic/base.h>

// constructor for dfd spline selection class

DFDSplineSelection::DFDSplineSelection(BSplineSelection* ss) {
    splsel = ss;
    txtsel = nil;
    latsel = nil;
}

// destructor for dfd spline selection class
DFDSplineSelection::~DFDSplineSelection() {
    /* delete splsel;
     delete txtsel; */
}

// set stored spline selection equal to given spline selection

void DFDSplineSelection::SetSplineSelection(BSplineSelection* ss) {
    splsel = ss;
}

// return stored spline selection

BSplineSelection* DFDSplineSelection::GetSplineSelection() {
    return splsel;
}

// set stored text selection (label) equal to given text selection

void DFDSplineSelection::SetTextSelection(TextSelection* ts) {
    txtsel = ts;
}

// return stored text selection (label)

```

```

TextSelection* DFDSplineSelection::GetTextSelection() {
    return txtsel;
}

// set stored text selection (latency) equal to given text selection

void DFDSplineSelection::SetLatencySelection(TextSelection* ts) {
    latsel = ts;
}

// return stored text selection (latency)

TextSelection* DFDSplineSelection::GetLatencySelection() {
    return latsel;
}

```

```

// file      sloperator.h
// description: Class description for OperatorSelection class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add definition of operator selection class.
 * This header file was specifically created for graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 30, 1990
 */

#ifndef sloperator_h
#define sloperator_h

#include <InterViews/defs.h>

class BSplineSelection;
// class DFDLineSelList;
// class DFDLineSelNode;
class DFDSplineSelList;
class DFDSplineSelNode;
class EllipseSelection;
// class LineSelection;
class TextBuffer;
class TextSelection;

class OperatorSelection {
public:
    OperatorSelection(EllipseSelection*);
    ~OperatorSelection();
    void SetEllipseSelection(EllipseSelection*);
    EllipseSelection* GetEllipseSelection();
    void SetTextSelection(TextSelection*);
    TextSelection* GetTextSelection();
    void SetMETSelection(TextSelection*);
    TextSelection* GetMETSelection();
// void InputDataFlowLineAppend(DFDLineSelNode*);
// void OutputDataFlowLineAppend(DFDLineSelNode*);
    void InputDataFlowSplineAppend(DFDSplineSelNode*);
    void OutputDataFlowSplineAppend(DFDSplineSelNode*);
    void SelfLoopAppend(DFDSplineSelNode*);
// char* FindLineInList(TextSelection*, LineSelection*);
    char* FindSplineInList(TextSelection*, BSplineSelection*);
// DFDLineSelList* GetInputDFLineList();
// DFDLineSelList* GetOutputDFLineList();
    DFDSplineSelList* GetInputDFSplineList();
    DFDSplineSelList* GetOutputDFSplineList();
    DFDSplineSelList* GetSelfLoopList();
// boolean FoundLineInLists(LineSelection*, LineSelection*);
    boolean FoundSplineInLists(BSplineSelection*, BSplineSelection*);
    boolean FoundSelfLoopInList(BSplineSelection*, BSplineSelection*);

```

```

char* FoundTextInLists(TextSelection*, TextSelection*);
void FoundSecondTextInLists(char*, TextSelection*);
char* GetPSDLText();
// boolean FindLineInSecondList(char*, char*, LineSelection*);
boolean FindSplineInSecondList(char*, char*, BSplineSelection*);
void SetPSDLText(char*);
boolean AddLatencyToSplineInList(TextSelection*, BSplineSelection*);

void RemoveInputFromPSDL(TextSelection*);
void RemoveOutputFromPSDL(TextSelection*);
void RemoveStateFromPSDL(TextSelection*);

protected:
void AddOperatorIdToPSDL(char*);
void AddInputToPSDL();
void AddOutputToPSDL();
void ReplaceInputStringInPSDL(char*, char*);
void ReplaceOutputStringInPSDL(char*, char*);
int FindIndex(char**, int);
void ReplaceStateStringInPSDL(char*, char*);
void AddStateToPSDL();
void AddMETToPSDL(TextSelection*);

EllipseSelection* elsel; // operator drawn in DFD
TextBuffer* txb; // buffer containing PSDL
TextSelection* txtsel; // operator's label
TextSelection* metsel; // operator's MET
// DFDLineSelList* input_df_line_list; // list of input data
// // flow lines
// DFDLineSelList* output_df_line_list; // list of output data
// // flow lines
DFDSplineSelList* input_df_spline_list; // list of input data
// // flow splines
DFDSplineSelList* output_df_spline_list; // list of output data
// // flow splines
DFDSplineSelList* selfloop_list; // list of self loops
};

#endif

```

```

// file          sloperator.c
// description:   Implementation of OperatorSelection class.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Add implmentation of operator selection class. Each operator selection
 * includes a pointer to the ellipse selection that represents the oper-
 * ator,
 * its label, all input and output dataflows and selfloops, and a text
 * buffer
 * containing all of the generated PSDL text.
 * This file was created specifically for the graphic editor.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  August 30, 1990
 */

#include "dfd_defs.h"
#include "dfdc_classes.h"
#include "dfdsplinelist.h"
#include "istring.h"
#include "sldfdspine.h"
#include "sllellipses.h"
#include "sloperator.h"
#include "slsplines.h"
#include "sltext.h"
#include <InterViews/textbuffer.h>
#include <InterViews/Graphic/base.h>
#include <InterViews/regexp.h>
#include <InterViews/Std/string.h>
#include <InterViews/Std/stdio.h>
#include <InterViews/Std/stdlib.h>

// constructor for operator selection class

OperatorSelection::OperatorSelection(EllipseSelection* es) {
    elsel = es;
    txtsel = nil;      // initially, no label is attached to operator
    metsel = nil;      // initially, no MET is attached to an operator
    // input_df_line_list = new DFDFLineSelList();
    // output_df_line_list = new DFDFLineSelList();
    input_df_spline_list = new DFDSplineSelList();
    output_df_spline_list = new DFDSplineSelList();
    selfloop_list = new DFDSplineSelList();
    char* op_string = new char [TXTBUFLLEN]; // create initial PSDL string
    strcpy(op_string, OPER_TKN);
    strcat(op_string, ID_TKN);
    strcat(op_string, "\n");
    strcat(op_string, SPEC_TKN);
    strcat(op_string, DESC_TKN);
    strcat(op_string, TEXT_TKN);
    strcat(op_string, END_TKN);
}

```

```

        txb = new TextBuffer(op_string, strlen(op_string), TXTBUFLLEN);
    }

    // destructor for operator selection class
    OperatorSelection::~~OperatorSelection() {
        delete elsel;
        delete txtsel;
        delete metsel;
        // delete input_df_line_list;
        // delete output_df_line_list;
        delete input_df_spline_list;
        delete output_df_spline_list;
        delete selfloop_list;
    }

    // Set stored ellipse selection with given ellipse selection

    void OperatorSelection::SetEllipseSelection(EllipseSelection* es) {
        elsel = es;
    }

    // Return stored ellipse selection

    EllipseSelection* OperatorSelection::GetEllipseSelection() {
        return elsel;
    }

    // Set stored text selection (label) with given text selection

    void OperatorSelection::SetTextSelection(TextSelection* ts) {

        // add operator's label to PSDL

        char* ts_string;
        if (ts != nil) {
            const char* copy_string;
            int len;
            copy_string = ts->GetOriginal(len);
            ts_string = new char[len+1];
            strncpy(ts_string, copy_string, len);
            ts_string[len] = '\0';
        }
        else {
            ts_string = ID_TKN;
        }
        AddOperatorIdToPSDL(ts_string);

        txtsel = ts;
    }

    // return stored text selection (label)

```



```

TextSelection* OperatorSelection::GetTextSelection() {
    return txtsel;
}
// Lines no longer used, use splines instead

/* ***** Start of Commented Out Code *****

// append given dfd line to list of input data flow lines

void OperatorSelection::InputDataFlowLineAppend(DFDLineSelNode* dfdlsn)
{
    AddInputToPSDL();
    input_df_line_list->Append(dfdlsn);
}

// append given dfd line to list of output data flow lines

void OperatorSelection::OutputDataFlowLineAppend(DFDLineSelNode* dfdlsn)
{
    AddOutputToPSDL();
    output_df_line_list->Append(dfdlsn);
}
***** End of Commented Out Code ***** */

// append given dfd spline to list of input data flow splines

void OperatorSelection::InputDataFlowSplineAppend(DFDSplineSelNode*
dfdssn) {
    AddInputToPSDL();
    input_df_spline_list->Append(dfdssn);
}

// append given dfd spline to list of output data flow splines

void OperatorSelection::OutputDataFlowSplineAppend(DFDSplineSelNode*
dfdssn) {
    AddOutputToPSDL();
    output_df_spline_list->Append(dfdssn);
}

// append given dfd spline to list of self loops

void OperatorSelection::SelfLoopAppend(DFDSplineSelNode* dfdssn) {
    AddStateToPSDL();
    selfloop_list->Append(dfdssn);
}

/* ***** Start of Commented Out Code *****

// search the lists of input and output data flow lines to find given
// line selection. If found, replace its label with given text selection.

```

```

// Return the name of the string in PSDL for the former label;

char* OperatorSelection::FindLineInList(TextSelection* ts,
                                         LineSelection* ls) {

    const char* copy_string;
    char* old_string = nil;
    char* new_string;
    TextSelection* oldts;
    input_df_line_list->SetCur(ls);
    if (!input_df_line_list->AtEnd()) {
        oldts = input_df_line_list->GetCur()->GetSelection()->
            GetTextSelection();

        if (oldts == nil) {
            old_string = "<input id>";
        }
        else {
            int len;
            copy_string = oldts->GetOriginal(len);
            old_string = new char[len+1];
            strncpy(old_string, copy_string, len);
            old_string[len] = '\0';
        }
        int len1;
        copy_string = ts->GetOriginal(len1);
        new_string = new char[len1 + 1];
        strncpy(new_string, copy_string, len1);
        new_string[len1] = '\0';
        printf("new string is %s\n", new_string);
        ReplaceInputStringInPSDL(old_string, new_string);
        input_df_line_list->GetCur()->GetSelection()->SetTextSelec-
tion(ts);
    }
    else {
        output_df_line_list->SetCur(ls);
        if (!output_df_line_list->AtEnd()) {
            oldts = output_df_line_list->GetCur()->GetSelection()
                ->GetTextSelection();

            if (oldts == nil) {
                old_string = "<output id>";
            }
            else {
                int len;
                copy_string = oldts->GetOriginal(len);
                old_string = new char[len + 1];
                strncpy(old_string, copy_string, len);
                old_string[len] = '\0';
            }
            int len1;
            copy_string = ts->GetOriginal(len1);
            new_string = new char[len1 + 1];
            strncpy(new_string, copy_string, len1);
            printf("new string is %s\n", new_string);
        }
    }
}

```

```

        new_string[len1] = '\0';
        ReplaceOutputStringInPSDL(old_string, new_string);
        output_df_line_list->GetCur()->GetSelection()->SetTextSelection(ts);
    }
}
if (old_string == "<output id>") {
    old_string = "<input id>";
}
else {
    if (old_string == "<input id>") {
        old_string = "<output id>";
    }
}
return old_string;
}

***** End of Commented Out Code ***** */

// Search lists of self loops, input and output data flow splines to find
// given spline selection. If found, replace its label with given text
// selection. Return the string in PSDL for the former label.

char* OperatorSelection::FindSplineInList(TextSelection* ts,
                                           BSplineSelection* ss) {
    char* old_string = nil;
    const char* copy_string;
    char* new_string = nil;
    TextSelection* oldts;
    selfloop_list->SetCur(ss);
    if (!selfloop_list->AtEnd()) {
        oldts = selfloop_list->GetCur()->GetSelection()
            ->GetTextSelection();
        if (oldts == nil) {
            old_string = ID_TKN;
        }
        else {
            int len;
            copy_string = oldts->GetOriginal(len);
            old_string = new char[len+1];
            strncpy(old_string, copy_string, len);
            old_string[len] = '\0';
        }
        int len7;
        copy_string = ts->GetOriginal(len7);
        new_string = new char[len7 + 1];
        strncpy(new_string, copy_string, len7);
        new_string[len7] = '\0';
        ReplaceStateStringInPSDL(old_string, new_string);
        selfloop_list->GetCur()->GetSelection()->SetTextSelection(ts);
    }
    else {
        input_df_spline_list->SetCur(ss);
    }
}

```

```

if (!input_df_spline_list->AtEnd()) {
    oldts = input_df_spline_list->GetCur()->GetSelection()
        ->GetTextSelection();
    if (oldts == nil) {
        old_string = ID_TKN;
    }
    else {
        int len;
        copy_string = oldts->GetOriginal(len);
        old_string = new char [len+1];
        strncpy(old_string,copy_string,len);
        old_string[len] = '\0';
    }
    int len1;
    copy_string = ts->GetOriginal(len1);
    new_string = new char[len1 + 1];
    strncpy(new_string,copy_string,len1);
    new_string[len1] = '\0';
    ReplaceInputStringInPSDL(old_string, new_string);
    input_df_spline_list->GetCur()->GetSelection()->
        SetTextSelection(ts);
}
else {
    output_df_spline_list->SetCur(ss);
    if (!output_df_spline_list->AtEnd()) {
        oldts = output_df_spline_list->GetCur()->GetSelection()
            ->GetTextSelection();
        if (oldts == nil) {
            old_string = ID_TKN;
        }
        else {
            int len;
            copy_string = oldts->GetOriginal(len);
            old_string = new char [len + 1];
            strncpy(old_string,copy_string,len);
            old_string[len] = '\0';
        }
        int len1;
        copy_string = ts->GetOriginal(len1);
        new_string = new char[len1 + 1];
        strncpy(new_string,copy_string,len1);
        new_string[len1] = '\0';
        ReplaceOutputStringInPSDL(old_string, new_string);
        output_df_spline_list->GetCur()->GetSelection()->
            SetTextSelection(ts);
    }
}
return old_string;
}

/* ***** Start of Commented Out Code *****

```

```

// Return stored list of input data flow lines

DFDLineSelList* OperatorSelection::GetInputDFLineList() {
    return input_df_line_list;
}

// Return stored list of output data flow lines

DFDLineSelList* OperatorSelection::GetOutputDFLineList() {
    return output_df_line_list;
}

***** End of Commented Out Code ***** */

// Return stored list of input data flow splines

DFDSplineSelList* OperatorSelection::GetInputDFSplineList() {
    return input_df_spline_list;
}

// Return stored list of output data flow splines

DFDSplineSelList* OperatorSelection::GetOutputDFSplineList() {
    return output_df_spline_list;
}

// Return stored list of self loops

DFDSplineSelList* OperatorSelection::GetSelfLoopList() {
    return selfloop_list;
}

/* ***** Start of Commented Out Code *****

// Find given line (replacee) in list of input and output data flow
// lines. If found, replace it with replacer.

boolean OperatorSelection::FoundLineInLists(LineSelection* replacee,
                                             LineSelection* replacer) {
    boolean found = false;
    DFDLineSelList* ill = GetInputDFLineList();
    for (ill->First(); !ill->AtEnd() && !found; ill->Next()) {
        if (ill->GetCur()->GetSelection()->GetLineSelection() == replacee)
        {
            found = true;
            ill->GetCur()->GetSelection()->SetLineSelection(replacer);
        }
    }

    DFDLineSelList* oll = GetOutputDFLineList();
    for (oll->First(); !oll->AtEnd() && !found; oll->Next()) {
        if (oll->GetCur()->GetSelection()->GetLineSelection() == replacee)

```

```

    {
        found = true;
        o1l->GetCur()->GetSelection()->SetLineSelection(replacer);
    }
    }
    return found;
}

***** End of Commented Out Code ***** */

// Search for given spline selection (replacee) in list of input and output
// data flow splines. If found, replace it with replacer.

boolean OperatorSelection::FoundSplineInLists(BSplineSelection* replacee,
                                              BSplineSelection* replacer) {
    boolean found = false;
    DFDSplineSelList* isl = GetInputDFSplineList();
    for (isl->First(); !isl->AtEnd() && !found; isl->Next()) {
        if (isl->GetCur()->GetSelection()->GetSplineSelection() == replacee) {
            found = true;
            isl->GetCur()->GetSelection()->SetSplineSelection(replacer);
        }
    }

    DFDSplineSelList* osl = GetOutputDFSplineList();
    for (osl->First(); !osl->AtEnd() && !found; osl->Next()) {
        if (osl->GetCur()->GetSelection()->GetSplineSelection() == replacee) {
            found = true;
            osl->GetCur()->GetSelection()->SetSplineSelection(replacer);
        }
    }
    return found;
}

// Search for given spline selection (replacee) in list of self loops.
// If found, replace it with replacer.

boolean OperatorSelection::FoundSelfLoopInList(BSplineSelection* replacee,
                                              BSplineSelection* replacer) {
    boolean found = false;
    DFDSplineSelList* sll = GetSelfLoopList();
    for (sll->First(); !sll->AtEnd() && !found; sll->Next()) {
        if (sll->GetCur()->GetSelection()->GetSplineSelection() == replacee) {
            found = true;
            sll->GetCur()->GetSelection()->SetSplineSelection(replacer);
        }
    }
    return found;
}

```

```

}

// Search for given text selection (replacee) in all lists attached
// to operator.  If found, replace it with replacer.

char* OperatorSelection::FoundTextInLists(TextSelection* replacee,
                                           TextSelection* replacer) {
    boolean found = false;
    int len;
    const char* tmp = replacee->GetOriginal(len);
    char* old_string = new char[len+1];
    strncpy(old_string,tmp,len);
    old_string[len] = '\0';
    char* new_string;
    if (replacer != nil) {
        tmp = replacer->GetOriginal(len);
        new_string = new char[len+1];
        strncpy(new_string,tmp,len);
        new_string[len] = '\0';
    }
    else {
        new_string = ID_TKN;
    }

/* ***** Start of Commented Out Code *****

    for (input_df_line_list->First(); !input_df_line_list->AtEnd() &&
!found;
        input_df_line_list->Next()) {
        if (input_df_line_list->GetCur()->GetSelection()->GetTextSelection()
            == replacee) {
            found = true;
            input_df_line_list->GetCur()->GetSelection()->
                SetTextSelection(replacer);
        }
    }

    for (output_df_line_list->First(); !output_df_line_list->AtEnd() &&
!found;
        output_df_line_list->Next()) {
        if (output_df_line_list->GetCur()->GetSelection()->GetTextSelection()
            == replacee) {
            found = true;
            output_df_line_list->GetCur()->GetSelection()->
                SetTextSelection(replacer);
        }
    }

***** End of Commented Out Code ***** */
    ClassId cid = replacee->GetClassId();
    switch (cid) {

```

```

case LABEL_DF:
    for (input_df_spline_list->First(); !input_df_spline_list->AtEnd()
        && !found; input_df_spline_list->Next()) {
        if (input_df_spline_list->GetCur()->GetSelection()
            ->GetTextSelection() == replacee) {
            found = true;
            ReplaceInputStringInPSDL(old_string,new_string);
            input_df_spline_list->GetCur()->GetSelection()->
                SetTextSelection(replacer);
        }
    }

    for (output_df_spline_list->First(); !output_df_spline_list-
>AtEnd()
        && !found; output_df_spline_list->Next()) {
        if (output_df_spline_list->GetCur()->GetSelection()
            ->GetTextSelection() == replacee) {
            found = true;
            ReplaceOutputStringInPSDL(old_string,new_string);
            output_df_spline_list->GetCur()->GetSelection()->
                SetTextSelection(replacer);
        }
    }
    if (found) {
        return old_string;
    }
    else {
        return nil;
    }
    break;
case LAT_DF:
    for (input_df_spline_list->First(); !input_df_spline_list->AtEnd()
        && !found; input_df_spline_list->Next()) {
        if (input_df_spline_list->GetCur()->GetSelection()
            ->GetLatencySelection() == replacee) {
            found = true;
            input_df_spline_list->GetCur()->GetSelection()->
                SetLatencySelection(replacer);
        }
    }

    for (output_df_spline_list->First(); !output_df_spline_list-
>AtEnd()
        && !found; output_df_spline_list->Next()) {
        if (output_df_spline_list->GetCur()->GetSelection()
            ->GetLatencySelection() == replacee) {
            found = true;
            output_df_spline_list->GetCur()->GetSelection()->
                SetLatencySelection(replacer);
        }
    }
    if (found) {

```



```

        return old_string;
    }
    else {
        return nil;
    }
    break;
case LABEL_SL:
    for (selfloop_list->First(); !selfloop_list->AtEnd() && !found;
        selfloop_list->Next()) {
        if (selfloop_list->GetCur()->GetSelection()->GetTextSelection()
            == replacee) {
            found = true;
            ReplaceStateStringInPSDL(old_string,new_string);
            selfloop_list->GetCur()->GetSelection()->
                SetTextSelection(replacer);
        }
    }
    if (found) {
        return old_string;
    }
    else {
        return nil;
    }
    break;
}
return nil;
}

```

// to update the PSDL for the second operator the data flow is attached to,
// this function will update only the PSDL

```

void OperatorSelection::FoundSecondTextInLists(char* old_string,
                                                TextSelection* new_ts) {
    boolean found = false;
    int len;
    char* new_string;
    if (new_ts != nil) {
        const char* tmp = new_ts->GetOriginal(len);
        new_string = new char[len+1];
        strncpy(new_string,tmp,len);
        new_string[len] = '\0';
    }
    else {
        new_string = ID_TKN;
    }

    for (input_df_spline_list->First(); !input_df_spline_list->AtEnd();
        input_df_spline_list->Next()) {
        if (input_df_spline_list->GetCur()->GetSelection()
            ->GetTextSelection() == new_ts) {
            found = true;

```

```

        ReplaceInputStringInPSDL(old_string,new_string);
    }
}

for (output_df_spline_list->First(); !output_df_spline_list->AtEnd()
    && !found; output_df_spline_list->Next()) {
    if (output_df_spline_list->GetCur()->GetSelection()
        ->GetTextSelection() == new_ts) {
        found = true;
        ReplaceOutputStringInPSDL(old_string,new_string);
    }
}

// Add operator id (label) to the PSDL that represents the operator

void OperatorSelection::AddOperatorIdToPSDL(char* ts_string) {
    int index = txb->ForwardSearch(new Regexp(OPER_TKN),0);
    if (index >= 0) {

        //      delete stored label and replace it with new label

        int end_index = txb->EndOfLine(index);
        txb->Delete(index,end_index-index);
        char* fixed_string = RemoveBadChars(ts_string);
        txb->Insert(index,fixed_string,strlen(fixed_string));
    }
}

// Extract PSDL text from buffer

char* OperatorSelection::GetPSDLText() {
    const char* text = txb->Text();
    int len = txb->Length();
    char* result = new char[len + 1];
    strncpy(result,text,len);
    result[len] = '\0';
    return result;
}

// Add input parameter of operator to psdl

void OperatorSelection::AddInputToPSDL() {
    char* spec_string;
    int index = txb->ForwardSearch(new Regexp(INPUT_SCH_TKN),0);
    if (index < 0) {

        //      INPUT not found, place after SPECIFICATION

        int index1 = txb->ForwardSearch(new Regexp(SPEC_SCH_TKN),0);
        if (index1 >= 0) {
            spec_string = new char[strlen(INPUT_TKN) +

```

```

                                strlen(TYPE_DECL_TKN) + 2];
strcpy(spec_string, INPUT_TKN);
strcat(spec_string, TYPE_DECL_TKN);
strcat(spec_string, "\n");
txb->Insert(index1 + 1, spec_string, strlen(spec_string));
    }
}
else {
    spec_string = new char[strlen(TYPE_DECL_TKN) + 3];
    strcpy(spec_string, TYPE_DECL_TKN);
    strcat(spec_string, ", \n");
    txb->Insert(index + 1, spec_string, strlen(spec_string));
}
}

// add output parameter to operator's psdl

void OperatorSelection::AddOutputToPSDL() {
    char* spec_string;
    int index = txb->ForwardSearch(new Regexp(OUTPUT_SCH_TKN), 0);
    if (index < 0) {

//        OUTPUT not found, place after INPUT

        int index1 = txb->Search(new Regexp(INPUT_SCH_TKN), 0, TXTBUFLen,
                                TXTBUFLen);
        if (index1 < 0) {

//            INPUT not found, place after SPECIFICATION

            int index2 = txb->ForwardSearch(new Regexp(SPEC_SCH_TKN), 0);
            if (index2 >= 0) {
                spec_string = new char[strlen(OUTPUT_TKN) +
                                        strlen(TYPE_DECL_TKN) + 2];
                strcpy(spec_string, OUTPUT_TKN);
                strcat(spec_string, TYPE_DECL_TKN);
                strcat(spec_string, "\n");
                txb->Insert(index2 + 1, spec_string, strlen(spec_string));
            }
        }
        else {

//            found INPUT, have to find proper place to put OUTPUT after that

            char* string_array[] = {GEN_SCH_TKN, STATES_SCH_TKN, EX-
CEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN, MRT_SCH_TKN,
                                KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                                END_SCH_TKN};
            int index3 = FindIndex(string_array, 10);
            if (index3 >= 0) {
                spec_string = new char[strlen(OUTPUT_TKN) +

```

```

        strlen(TYPE_DECL_TKN) + 2];
        strcpy(spec_string, OUTPUT_TKN);
        strcat(spec_string, TYPE_DECL_TKN);
        strcat(spec_string, "\n");
        txb->Insert(index3, spec_string, strlen(spec_string));
    }
}
else {
    spec_string = new char[strlen(TYPE_DECL_TKN) + 3];
    strcpy(spec_string, TYPE_DECL_TKN);
    strcat(spec_string, ", \n");
    txb->Insert(index + 1, spec_string, strlen(spec_string));
}

// replace old string in input portion of the PSDL text buffer with
// the new string

void OperatorSelection::ReplaceInputStringInPSDL(char* old_string,
                                                char* new_string) {
    int start_index = txb->Search(new Regexp(INPUT_SCH_TKN), 0, TXTBUFLen,
                                TXTBUFLen);

    char* old_fixed_string = RemoveBadChars(old_string);
    if (start_index >= 0) {
        char* string_array[] = {OUTPUT_SCH_TKN, GEN_SCH_TKN, STATES_SCH_TKN,
                                EXCEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN, MRT_SCH_TKN,
                                KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                                END_SCH_TKN};
        int end_index = FindIndex(string_array, 11);
        if (end_index >= start_index) {
            int index = txb->Search(new Regexp(old_fixed_string), start_in-
dex,
                                end_index - start_index + 1, end_index);
            if (index >= 0) {
                txb->Delete(index, strlen(old_fixed_string));
                char* fixed_string = RemoveBadChars(new_string);
                txb->Insert(index, fixed_string, strlen(fixed_string));
            }
        }
    }

    // replace old string in output portion of the PSDL text buffer with
    // the new string

    void OperatorSelection::ReplaceOutputStringInPSDL(char* old_string,
                                                    char* new_string) {
        int start_index = txb->Search(new Regexp(OUTPUT_SCH_TKN), 0, TXTBU-
FLEN,
                                TXTBUFLen);

        char* old_fixed_string = RemoveBadChars(old_string);

```

```

if (start_index >= 0) {
    char* string_array[] = {GEN_SCH_TKN, STATES_SCH_TKN,
                            EXCEPT_SCH_TKN,
                            MET_SCH_TKN, MCP_SCH_TKN, MRT_SCH_TKN,
                            KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                            END_SCH_TKN};
    int end_index = FindIndex(string_array, 10);
    if (end_index >= start_index) {
        int index = txb->Search(new Regexp(old_fixed_string), start_in-
dex,
                                end_index - start_index + 1, end_index);
        if (index >= 0) {
            txb->Delete(index, strlen(old_fixed_string));
            char* fixed_string = RemoveBadChars(new_string);
            txb->Insert(index, fixed_string, strlen(fixed_string));
        }
    }
}

/* ***** Start of Commented Out Code *****

// find second occurrence of line in the list in order to properly update
// the operator's PSDL

boolean OperatorSelection::FindLineInSecondList(char* old_string,
                                                char* new_string,
                                                LineSelection* ls) {
    input_df_line_list->SetCur(ls);
    if (!input_df_line_list->AtEnd()) {
        printf("new_string is %s\n", new_string);
        ReplaceInputStringInPSDL(old_string, new_string);
        return true;
    }
    else {
        output_df_line_list->SetCur(ls);
        if (!output_df_line_list->AtEnd()) {
            printf("new string is %s\n", new_string);
            ReplaceOutputStringInPSDL(old_string, new_string);
            return true;
        }
    }
    return false;
}

/* ***** End of Commented Out Code ***** */

// find second occurrence of spline in the list in order to properly update
// the operator's PSDL

boolean OperatorSelection::FindSplineInSecondList(char* old_string,
                                                char* new_string,
                                                BSplineSelection* ss) {

```

```

        input_df_spline_list->SetCur(ss);
        if (!input_df_spline_list->AtEnd()) {
            ReplaceInputStringInPSDL(old_string,new_string);
            return true;
        }
        else {
            output_df_spline_list->SetCur(ss);
            if (!output_df_spline_list->AtEnd()) {
                ReplaceOutputStringInPSDL(old_string,new_string);
                return true;
            }
        }
        return false;
    }

void OperatorSelection::SetMETSelection(TextSelection* met_ts) {
    AddMETToPSDL(met_ts);
    metssel = met_ts;
}

// return stored maximum execution time

TextSelection* OperatorSelection::GetMETSelection() {
    return metssel;
}

// search through text buffer to find place where one of the string array
// elements are located

int OperatorSelection::FindIndex(char** string_array, int size) {
    int index;
    for (int i = 0; i < size; ++i) {
        index = txb->Search(new Regexp(string_array[i]), 0, TXTBUFLen,
                                TXTBUFLen);

        if (index >= 0) {
            return index;
        }
    }
    return -1;
}

// set psdl text buffer to contents of given string

void OperatorSelection::SetPSDLText(char* text) {
    if (txb != nil)
        delete txb;
    txb = new TextBuffer(text, strlen(text), TXTBUFLen);
}

// add the latency of the data flow to the data flow found in the input
// or output list

```

```

boolean OperatorSelection::AddLatencyToSplineInList(TextSelection* la-
tency,
                                BSplineSelection* ss) {
    for (input_df_spline_list->First(); !input_df_spline_list->AtEnd();
         input_df_spline_list->Next()) {
        if (input_df_spline_list->GetCur()->GetSelection()
            ->GetSplineSelection() == ss) {
            input_df_spline_list->GetCur()->GetSelection()
                ->SetLatencySelection(latency);
            return true;
        }
    }
    for (output_df_spline_list->First(); !output_df_spline_list->AtEnd();
         output_df_spline_list->Next()) {
        if (output_df_spline_list->GetCur()->GetSelection()
            ->GetSplineSelection() == ss) {
            output_df_spline_list->GetCur()->GetSelection()
                ->SetLatencySelection(latency);
            return true;
        }
    }
    return false;
}

// add state parameter to operator's psdl

void OperatorSelection::AddStateToPSDL() {
    char* spec_string;
    char* string_array[] = {GEN_SCH_TKN, EXCEPT_SCH_TKN,
                            MET_SCH_TKN, MCP_SCH_TKN, MRT_SCH_TKN,
                            KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                            END_SCH_TKN};
    int index3 = FindIndex(string_array, 9);
    if (index3 >= 0) {
        txb->Insert(index3, ST_TKN, strlen(ST_TKN));
    }
}

// replace old string in state portion of the PSDL text buffer with
// the new string

void OperatorSelection::ReplaceStateStringInPSDL(char* old_string,
                                                  char* new_string) {
    int start_index = txb->Search(new Regexp(STATES_SCH_TKN), 0, TXTBU-
FLEN,
                                TXTBUFLEN);
    if (start_index >= 0) {
        char* string_array[] = {GEN_SCH_TKN, EXCEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN, MRT_SCH_TKN,
                                KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                                END_SCH_TKN};
        int end_index = FindIndex(string_array, 9);
    }
}

```

```

        if (end_index >= start_index) {
            char* old_fixed_string = RemoveBadChars(old_string);
            int index = txb->Search(new Regexp(old_fixed_string), start_in-
dex,
                                end_index - start_index + 1, end_index);
            if (index >= 0) {
                txb->Delete(index, strlen(old_fixed_string));
                char* fixed_string = RemoveBadChars(new_string);
                txb->Insert(index, fixed_string, strlen(fixed_string));
            }
        }
    }

// add MET parameter to operator's psdl

void OperatorSelection::AddMETToPSDL(TextSelection* ts) {
    char* new_string;
    if (ts != nil) {
        int len;
        const char* temp_string = ts->GetOriginal(len);
        new_string = new char[len + 1];
        strncpy(new_string, temp_string, len);
        new_string[len] = '\0';
    }
    int index = txb->ForwardSearch(new Regexp(MET_SCH_TKN), 0);
    if (ts != nil) {
        if (index < 0) {
            char* spec_string;
            char* string_array[] = {MCP_SCH_TKN, MRT_SCH_TKN,
                                   KEY_SCH_TKN, DESC_SCH_TKN, AX_SCH_TKN,
                                   END_SCH_TKN};
            int index3 = FindIndex(string_array, 6);
            if (index3 >= 0) {
                spec_string = new char[strlen(MET_TKN) +
                                           strlen(new_string) + 2];
                strcpy(spec_string, MET_TKN);
                strcat(spec_string, new_string);
                strcat(spec_string, "\n");
                txb->Insert(index3, spec_string, strlen(spec_string));
            }
        }
        else {
            int end_index = txb->EndOfLine(index);
            txb->Delete(index, end_index - index);
            txb->Insert(index, new_string, strlen(new_string));
        }
    }
    else {
        if (index >= 0) {
            int start_line_index = txb->BeginningOfLine(index);
            int end_line_index = txb->EndOfLine(index);

```



```

        txb->Delete(start_line_index, end_line_index - start_line_index
                    + 1);
    }
}

// Remove input type declaration from PSDL

void OperatorSelection::RemoveInputFromPSDL(TextSelection* ts) {
    char* id;
    if (ts != nil) {
        int len;
        const char* tmp = ts->GetOriginal(len);
        id = new char[len + 1];
        strncpy(id, tmp, len);
        id[len] = '\0';
    }
    else {
        id = ID_TKN;
    }
    char* fixed_id = RemoveBadChars(id);
    int start_index = txb->Search(new Regexp(INPUT_SCH_TKN), 0,
                                TXTBUFLen, TXTBUFLen);

    if (start_index != 0) {
        char* string_array[] = {OUTPUT_SCH_TKN, GEN_SCH_TKN,
                                STATES_SCH_TKN, EXCEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN,
                                MRT_SCH_TKN, KEY_SCH_TKN,
                                DESC_SCH_TKN, AX_SCH_TKN,
                                END_SCH_TKN};

        int end_index = FindIndex(string_array, 11);
        if (end_index > start_index) {
            int index = txb->Search(new Regexp(fixed_id), start_index,
                                    end_index - start_index + 1,
                                    end_index);

            if (index >= 0) {
                int end_line_index = txb->EndOfLine(index);
                int start_line_index = txb->BeginningOfLine(index);
                txb->Delete(start_line_index,
                            end_line_index - start_line_index + 1);
            }
        }
    }
}

// Remove output type declaration from PSDL

void OperatorSelection::RemoveOutputFromPSDL(TextSelection* ts) {
    char* id;
    if (ts != nil) {
        int len;
        const char* tmp = ts->GetOriginal(len);
    }
}

```

```

        id = new char[len + 1];
        strncpy(id,tmp,len);
        id[len] = '\0';
    }
    else {
        id = ID_TKN;
    }
    char* fixed_id = RemoveBadChars(id);
    int start_index = txb->Search(new Regexp(OUTPUT_SCH_TKN), 0,
                                  TXTBUFLen, TXTBUFLen);

    if (start_index >= 0) {
        char* string_array[] = {GEN_SCH_TKN,
                                STATES_SCH_TKN, EXCEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN,
                                MRT_SCH_TKN, KEY_SCH_TKN,
                                DESC_SCH_TKN, AX_SCH_TKN,
                                END_SCH_TKN};

        int end_index = FindIndex(string_array, 10);
        if (end_index > start_index) {
            int index = txb->Search(new Regexp(fixed_id), start_index,
                                      end_index - start_index + 1,
                                      end_index);

            if (index >= 0) {
                int end_line_index = txb->EndOfLine(index);
                int start_line_index = txb->BeginningOfLine(index);
                txb->Delete(start_line_index,
                          end_line_index - start_line_index + 1);
            }
        }
    }
}

// Remove state declaration from PSDL

void OperatorSelection::RemoveStateFromPSDL(TextSelection* ts) {
    char* id;
    if (ts != nil) {
        int len;
        const char* tmp = ts->GetOriginal(len);
        id = new char[len + 1];
        strncpy(id,tmp,len);
        id[len] = '\0';
    }
    else {
        id = ID_TKN;
    }
    char* fixed_id = RemoveBadChars(id);
    int start_index = txb->Search(new Regexp(STATES_SCH_TKN), 0,
                                  TXTBUFLen, TXTBUFLen);

    if (start_index >= 0) {
        char* string_array[] = {GEN_SCH_TKN, EXCEPT_SCH_TKN,
                                MET_SCH_TKN, MCP_SCH_TKN,

```

```

        MRT_SCH_TKN, KEY_SCH_TKN,
        DESC_SCH_TKN, AX_SCH_TKN,
        END_SCH_TKN);
int end_index = FindIndex(string_array, 9);
if (end_index > start_index) {
    int index = txb->Search(new Regexp(fixed_id), start_index,
        end_index - start_index + 1,
        end_index);

    if (index >= 0) {
        int end_line_index = txb->EndOfLine(index);
        int start_line_index = txb->BeginningOfLine(index);
        txb->Delete(start_line_index,
            end_line_index - start_line_index + 1);
    }
}
}
}

```

```

// file      tools.c
// description: Implementation of Tools class and its subclasses.

// $Header: tools.c,v 1.8 89/10/09 14:50:03 linton Exp $
// implements class Tools.

/* Changes made to conform Idraw to CAPS graphic editor:
 * Change ReshapeTool to be ModifyTool to be consistent with user's view.
 * Add AnnotateTool class to show annotation view of operator selected.
 * Add DecomposeTool class to decompose selected operator into sub operators.
 * Remove TextTool and replace it with CommentTool and LabelTool.
 * Remove StretchTool, MultiLineTool, PolygonTool, and ClosedBSplineTool,
 * ScaleTool, and MagnifyTool because they are not needed in a data flow
 * diagram editor.
 * Add new function to each subclass of IdrawTool (SetMessage) to set
 * the value of the message in the message block each time a tool is run.
 * Removed call to Panel's SetCur to highlight select as current tool.
 * Add METTool to add maximum execution time of operator.
 * Remove Line from the tools; Spline will be used to draw a line.
 * Change Annotate to Specify to make it clearer what the user is doing
 * and add streams and constraints to tools.
 * Add LatencyTool to add maximum firing time of data flow.
 *
 * Changes made by:   Mary Ann Cummings
 * Last change made:  October 21, 1990
 */

#include "editor.h"
#include "keystrokes.h"
#include "mapkey.h"
#include "tools.h"
#include <InterViews/box.h>
#include <InterViews/event.h>
#include <InterViews/painter.h>
#include <InterViews/shape.h>
#include <InterViews/Std/string.h>

// An IdrawTool enters itself into the MapKey so Idraw can send
// a KeyEvent to the right IdrawTool.

class IdrawTool : public PanelItem {
public:
    IdrawTool(Panel*, const char*, char, Editor*, MapKey*);
protected:
    Editor* editor;    // handles drawing and editing operations
};

// IdrawTool stores the editor pointer and enters itself and its
// associated character into the MapKey.

```

```

IdrawTool::IdrawTool (Panel* p, const char* n, char c, Editor* e,
MapKey* mapkey) : (p, n, mapkey->ToStr(c), c, e) {
    editor = e;
    mapkey->Enter(this, c);
}

// A SelectTool selects a set of Selections.

class SelectTool : public IdrawTool {
public:
    SelectTool (Panel* p, Editor* e, MapKey* mk)
    : (p, "Select", SELECTCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick object with LMB or hold down button to draw rectangle");
        strcat(msg,
            " around more than 1 object");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleSelect(e);
    }
};

// A MoveTool moves a set of Selections.

class MoveTool : public IdrawTool {
public:
    MoveTool (Panel* p, Editor* e, MapKey* mk)
    : (p, "Move", MOVECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick object with LMB");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleMove(e);
    }
};

// The following is not needed in a data flow diagram

/* ***** Start of Commented Out Code *****

// A ScaleTool scales a set of Selections.

class ScaleTool : public IdrawTool {
public:
    ScaleTool (Panel* p, Editor* e, MapKey* mk)
    : (p, "Scale", SCALECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,

```

```

        "pick object with LMB");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleScale(e);
    }
};

// A StretchTool stretches a set of Selections.

class StretchTool : public IdrawTool {
public:
    StretchTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Stretch", STRETCHCHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandleStretch(e);
    }
};

// A RotateTool rotates a set of Selections.

class RotateTool : public IdrawTool {
public:
    RotateTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Rotate", ROTATECHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandleRotate(e);
    }
};

***** End of Commented Out Code ***** */

// A Modify Tool modifies a Selection.

class ModifyTool : public IdrawTool {
public:
    ModifyTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Modify", MODIFYCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick data flow with LMB");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleModify(e);
    }
};

// The following code is not needed for a data flow diagram

/* ***** Start of Commented Out Code *****

// A MagnifyTool magnifies a part of the drawing.

```

```

class MagnifyTool : public IdrawTool {
public:
    MagnifyTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Magnify", MAGNIFYCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick object with LMB");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleMagnify(e);
    }
};

***** End of Commented Out Code ***** */

// A SpecifyTool opens up syntax directed editor for selected component
// of drawing to add PSDL specification

class SpecifyTool : public IdrawTool {
public:
    SpecifyTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Specify", SPECIFYCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick operator with LMB or click anywhere else for specification");
        strcat(msg,
            " of entire drawing");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleSpecify(e);
    }
};

// A StreamsTools opens up a syntax directed editor for drawing to add
// PSDL streams

class StreamsTool : public IdrawTool {
public:
    StreamsTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Streams", STREAMSCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "Use LMB to click anywhere in drawing");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleStreams(e);
    }
};

```

```
// A ConstraintsTools opens up a syntax directed editor for drawing to add
// PSDL constraints
```

```
class ConstraintsTool : public IdrawTool {
public:
    ConstraintsTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Constraints", CONSTRAINTSCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "Use LMB to click anywhere in drawing");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleConstraints(e);
    }
};
```

```
// A DecomposeTool will open new graphic editor for lower level of DFD
```

```
class DecomposeTool : public IdrawTool {
public:
    DecomposeTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Decompose", DECOMPOSECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick operator with LMB");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleDecompose(e);
    }
};
```

```
// A CommentTool draws some text.
```

```
class CommentTool : public IdrawTool {
public:
    CommentTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Comment", COMMENTCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "click outside of objects to add text");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleText(e);
    }
};
```

```
// A LabelTool draws some text for one of the components of the DFD
```



```

class LabelTool : public IdrawTool {
public:
    LabelTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Label", LABELCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick object with LMB, add text, then click outside objects");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleLabel(e);
    }
};

// A METTool adds the maximum execution of the operator to the drawing

class METTool : public IdrawTool {
public:
    METTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "MET", METCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick operator with LMB, add text, then click outside object");
        editor->ResetMessage(msg);
    }
    void Perform(Event& e) {
        editor->HandleMET(e);
    }
};

// A LatencyTool adds the latency time of the specified data flow to the
// drawing

class LatencyTool : public IdrawTool {
public:
    LatencyTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "Latency", LATENCYCHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "pick data flow with LMB, add text, then click outside object");
        editor->ResetMessage(msg);
    }
    void Perform(Event& e) {
        editor->HandleLatency(e);
    }
};

// Line no longer used, use spline to draw a line.

/* ***** Start of Commented Out Code *****

// A LineTool draws a line.

```

```

class LineTool : public IdrawTool {
public:
    LineTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", LINECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "click at stop to place one endpoint, hold down button and ");
        strcat(msg,
            "stretch line and release button at other endpoint");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleLine(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        Coord x0 = offx + side * 1/5;
        Coord y0 = offy + side * 4/5;
        Coord x1 = offx + side * 4/5;
        Coord y1 = offy + side * 1/5;
        output->Line(canvas, x0, y0, x1, y1);
    }
};

```

// The following is not needed for the data flow diagram

// A MultiLineTool draws a set of connected lines.

```

class MultiLineTool : public IdrawTool {
public:
    MultiLineTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", MULTILINECHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandleMultiLine(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        const int N = 4;
        Coord x[N];
        Coord y[N];
        x[0] = offx + side * 1/5;
        y[0] = offy + side * 4/5;
        x[1] = offx + side * 1/2;
        y[1] = offy + side * 4/5 - side * 1/10;
        x[2] = offx + side * 1/2;
        y[2] = offy + side * 1/5 + side * 1/10;
        x[3] = offx + side * 4/5;
        y[3] = offy + side * 1/5;
        output->MultiLine(canvas, x, y, N);
    }
};

```

```

    }
};

***** End of Commented Out Code ***** */

// A BSplineTool draws an open B-spline.

class BSplineTool : public IdrawTool {
public:
    BSplineTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", BSPLINECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "click LMB at each spot to place segment of spline");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleBSpline(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        const int N = 4;
        Coord x[N];
        Coord y[N];
        x[0] = offx + side * 1/5;
        y[0] = offy + side * 4/5;
        x[1] = offx + side * 1/2;
        y[1] = offy + side * 4/5;
        x[2] = offx + side * 1/2;
        y[2] = offy + side * 1/5;
        x[3] = offx + side * 4/5;
        y[3] = offy + side * 1/5;
        output->BSpline(canvas, x, y, N);
    }
};

// An EllipseTool draws an ellipse.

class EllipseTool : public IdrawTool {
public:
    EllipseTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", ELLIPSECHAR, e, mk) {}
    void SetMessage() {
        strcpy(msg,
            "click with LMB to draw operator centered at that point");
        editor->ResetMessage(msg);
    }
    void Perform (Event& e) {
        editor->HandleEllipse(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {

```

```

    IdrawTool::Redraw(l, b, r, t);
    Coord x0 = offx + side * 1/2;
    Coord y0 = offy + side * 1/2;
    Coord xradius = side * 1/3 + side * 1/16;
    Coord yradius = side * 1/3 - side * 1/16;
    output->Ellipse(canvas, x0, y0, xradius, yradius);
}
};

// The following is not needed for a data flow diagram

/* ***** Start of Commented Out Code *****

// A RectTool draws a rectangle.

class RectTool : public IdrawTool {
public:
    RectTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", RECTCHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandleRect(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        Coord x0 = offx + side * 1/5;
        Coord y0 = offy + side * 1/5;
        Coord x1 = offx + side * 4/5;
        Coord y1 = offy + side * 4/5;
        output->Rect(canvas, x0, y0, x1, y1);
    }
};

// A PolygonTool draws a polygon.

class PolygonTool : public IdrawTool {
public:
    PolygonTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", POLYGONCHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandlePolygon(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        const int N = 5;
        Coord x[N];
        Coord y[N];
        x[0] = cffx + side * 1/5 + side * 1/8;
        y[0] = offy + side * 1/5;
        x[1] = offx + side * 1/5;
        y[1] = offy + side * 1/2;

```

```

    x[2] = offx + side * 1/2;
    y[2] = offy + side * 4/5;
    x[3] = offx + side * 4/5;
    y[3] = offy + side * 1/2 + side * 1/8;
    x[4] = offx + side * 4/5 - side * 1/32;
    y[4] = offy + side * 1/5 + side * 1/8;
    output->Polygon(canvas, x, y, N);
}
};

// A ClosedBSplineTool draws a closed B-spline.

class ClosedBSplineTool : public IdrawTool {
public:
    ClosedBSplineTool (Panel* p, Editor* e, MapKey* mk)
        : (p, "", CLOSEDBSPLINECHAR, e, mk) {}
    void Perform (Event& e) {
        editor->HandleClosedBSpline(e);
    }
protected:
    void Redraw (Coord l, Coord b, Coord r, Coord t) {
        IdrawTool::Redraw(l, b, r, t);
        const int N = 6;
        Coord x[N];
        Coord y[N];
        x[0] = offx + side * 1/10;
        y[0] = offy + side * 1/2;
        x[1] = offx + side * 3/5;
        y[1] = offy + side * 1/5;
        x[2] = offx + side * 4/5;
        y[2] = offy + side * 2/5;
        x[3] = offx + side * 1/2;
        y[3] = offy + side * 1/2;
        x[4] = offx + side * 4/5;
        y[4] = offy + side * 3/5;
        x[5] = offx + side * 3/5;
        y[5] = offy + side * 4/5;
        output->ClosedBSpline(canvas, x, y, N);
    }
};

***** End of Commented Out Code ***** */

// Tools creates its tools.

Tools::Tools (Editor* e, MapKey* mk) {
    Init(e, mk);
}

// Handle tells one of the tools to perform its function if a
// DownEvent occurs.

void Tools::Handle (Event& e) {

```

```

        switch (e.eventType) {
        case DownEvent:
        switch (e.button) {
        case LEFTMOUSE:
            PerformCurrentFunction(e);
            break;
        case MIDDLEMOUSE:
            PerformTemporaryFunction(e, MOVECHAR);
            break;
        case RIGHTMOUSE:
            PerformTemporaryFunction(e, SELECTCHAR);
            break;
        default:
            break;
        }
        default:
        break;
    }
}

// Init creates the tools, lays them together, and inserts them.

void Tools::Init (Editor* e, MapKey* mk) {
    PanelItem* first = new SelectTool(this, e, mk);

    VBox* tools = new VBox;
    tools->Insert(first);
    tools->Insert(new MoveTool(this, e, mk));
    // tools->Insert(new ScaleTool(this, e, mk));

    // tools->Insert(new StretchTool(this, e, mk));
    // tools->Insert(new RotateTool(this, e, mk));

    tools->Insert(new ModifyTool(this, e, mk));
    // tools->Insert(new MagnifyTool(this, e, mk));
    tools->Insert(new SpecifyTool(this, e, mk));
    tools->Insert(new StreamsTool(this, e, mk));
    tools->Insert(new ConstraintsTool(this, e, mk));
    tools->Insert(new DecomposeTool(this, e, mk));
    tools->Insert(new CommentTool(this, e, mk));
    tools->Insert(new LabelTool(this, e, mk));
    tools->Insert(new METTool(this, e, mk));
    tools->Insert(new LatencyTool(this, e, mk));
    // tools->Insert(new LineTool(this, e, mk));

    // tools->Insert(new MultiLineTool(this, e, mk));

    tools->Insert(new BSplineTool(this, e, mk));
    tools->Insert(new EllipseTool(this, e, mk));

    // tools->Insert(new RectTool(this, e, mk));
    // tools->Insert(new PolygonTool(this, e, mk));

```

```
// tools->Insert(new ClosedBSplineTool(this, e, mk));

    Insert(tools);
// SetCur(first);
}

// Reconfig makes Tools's shape unstretchable but shrinkable.

void Tools::Reconfig () {
    Panel::Reconfig();
    shape->Rigid(0, 0, vfil, 0);
}
```

LIST OF REFERENCES

1. Raum, H. G., *Design and Implementation of an Expert User Interface for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.
2. Reiss, S. P., *GARDEN Tools: Support for Graphical Programming*, Advanced Programming Environments Proceedings, June 1986.
3. Jones, O., *Introduction to the X Window System*, Prentice Hall, 1989.
4. Linton, M. A., Vlissides, J. M., and Calder P. R., *Composing User Interfaces with Interviews*, IEEE Computer, February 1989.
5. Lippman, S. B., *C++ Primer*, Addison-Wesley, 1989.
6. Luqi, Berzins, V., and Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, October 1988.
7. Vlissides, J. M., and Linton, M. A., *Applying Object-Oriented Design to Structured Graphics*, Proceedings of the 1988 USENIX C++ Conference, October 1988.
8. Yourdon, E., *Modern Structured Analysis*, YOURDON Press, 1989.
9. Boar, B. H., *Application Prototyping: A Requirements Definition Strategy For the 80's*, John Wiley and Sons, Inc., 1984.
10. Berzins, V., and Luqi, *Rapidly Prototyping Real-Time Systems*, IEEE Software, September 1988.
11. Luqi, Barnes, P., and Zyda, M., *Graphical Tool for Computer-Aided Prototyping*, Information and Software Technology, Vol. 32, No. 3, April, 1990.
12. White, L. J., *The Development of a Rapid Prototyping Environment*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1989.
13. Linton, M. A., Calder, P. R., and Vlissides, J. M., *A C++ Graphical Interface Toolkit*, Technical Report CSL-TR-88-358, Stanford University, July 1988.

14. Jones, O., *Introduction to the X Window System*, Prentice Hall, 1989.
15. Sun, C. H., *Developing Portable User Interfaces for Ada Command Control Software*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1990.
16. Shneiderman, B., *Designing the User Interface*, Addison-Wesley, 1987.
17. Thorstenson, R. K., *A Graphical Editor for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.
18. *Dictionary of Computing*, Oxford University Press, 1990.
19. Coskun, V., and Kesoglu, C., *A Software Prototype for a Command, Control, Communications, and Intelligence (C3I) Workstation*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1990.

BIBLIOGRAPHY

Barnes, P. D., *A Decision-Based Methodology For Object Oriented-Design*, M.S. Thesis, AFIT/GCS/ENG/88D-1, Air Force Institute Of Technology, Wright-Patterson AFB, Ohio, December 1988.

Berzins, V., and Luqi, *Software with Abstractions*, Addison-Wesley, 1991.

Booch, G., *Software Engineering With Ada*, Benjamin/Cummings Publishing Company, Inc., 1987.

Brown, J. R., and Cunningham, S., *Programming The User Interface*, John Wiley & Sons, Inc., 1989.

Dumas, J. S., *Designing User Interfaces For Software*, Prentice Hall, 1988.

Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, May 1989.

Luqi, and Lee, Y., *Interactive Control of Prototyping Process*, Proceedings of COMPSAC 89, September, 1989.

INITIAL DISTRIBUTION LIST

<p>Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145</p>	<p>2</p>
<p>Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943</p>	<p>2</p>
<p>Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268</p>	<p>1</p>
<p>Director of Research Administrations Attn: Prof. Howard Code 012 Naval Postgraduate School Monterey, CA 93943</p>	<p>1</p>
<p>Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943</p>	<p>1</p>
<p>Office of Naval Technology Attn: CDR Jane Van Fossen, Code 227 800 North Quincy Street Arlington, VA 22217-5000</p>	<p>1</p>
<p>Prof. Luqi, Code CS/Lq Naval Postgraduate School Monterey, CA 93943</p>	<p>3</p>
<p>Naval Surface Warfare Center Attn: Philip Q. Hwang (U33) Silver Spring, MD 20903-5000</p>	<p>1</p>

Naval Surface Warfare Center 1
Attn: William McCoy (K53)
Dahlgren, VA 22448

Naval Surface Warfare Center 1
Technical Library
Dahlgren, VA 22448

InterViews 1
Center for Interactive Systems, Room 213
Stanford University
Stanford, CA 94305